Objective FP7-ICT-2009-5-257448/D-5.4

Future Networks

Project 257448

"SAIL – Scalable and Adaptable Internet Solutions"

D-5.4
(D.D.3) Refined Architecture

Date of preparation: **2012-10-31**
Start date of Project: **2010-08-01**
Project Coordinator: **Thomas Edwall**
Ericsson AB

Revision: **1.0**
Duration: **2013-01-31**

Document Properties

Document Number:	D-5.4
Document Title:	**(D.D.3) Refined Architecture**
Document Responsible:	Azimeh Sefidcon (EAB)
Document Editor:	Paul Murray (HP)
Authors:	Enrique Fernandez (EAB), Bob Melander (EAB), Hareesh Puthalath (EAB), Azimeh Sefidcon (EAB), Victor Souza (EAB), Ayush Sharma (FHG), Sathyanarayanan Rangarajan(FHG), Paul Murray (HP), Luis Vaquero (HP), Suksant Sae Lor (HP), Rolf Stadler (KTH), Fetahi Wuhib (KTH), Houssem Medhioub (IT), Marcio Melo (PTIN), Joao Soares (PTIN), Jorge Carapinha (PTIN), Bjorn Bjurling (SICS), Daniel Gillblad (SICS), Rebecca Steinert (SICS), Pedro A. Aranda (TID), Matthias Keller (UPB), Fabian Schneider (NEC), Daniel Turull (KTH), Thomas Begin (INRIA), Paulo Goncalves (INRIA), Avi Miron (Technion)
Target Dissemination Level:	PU
Status of the Document:	Final
Version:	1.0

Production Properties:

Reviewers:	Bengt Ahlgren (SICS), Hannu Flink (NSN), Benoit Tremblay (EAB)

Document History:

Revision	Date	Issued by	Description
1.0	2012-10-31	Paul Murray	Final Version

Disclaimer:

Abstract:

This document describes the final cloud networking architecture proposed by SAIL work package D. Cloud networking aims at providing on-demand elastic network services to connect existing data centre based cloud infrastructures across wide area networks. In order to achieve that we propose the flash network slice, a network resource that can be provisioned and reconfigured in a timeframe that is compatible with current provisioning of virtual machines in a data centre. We consider a multi-provider scenario, where network and data centre providers must cooperate to implement global virtual infrastructures. This is enabled through provisioning interfaces and inter-provider protocols defined in the present architecture. The architecture introduces a layering of functions, interfaces and interactions within single cloud network operators, across cloud network operators, and with cloud network customers to implement complete end-to-end services. Management algorithms for user goal translation, fault and resource management are presented, including results from practical experimentation and simulations. Security goals and policy based access control mechanisms are described. This final version of the cloud networking architecture represents a refinement over the initial architecture after experience gained in prototyping and simulating management and security functions and the implementation of a complete integrated multi-data centre, multi-operator cloud networking eco-system.

Keywords:

cloud computing, cloud networking, network virtualisation, infrastructure as a service, inter-provider virtual infrastructure, resource management, cloud network security

Executive Summary

This document is a public deliverable of the Scalable Adaptive Internet Solutions (SAIL) EU-FP7 project [1] and describes the Cloud Networking (CloNe) architecture. CloNe provides on-demand elastic network services to connect existing data centre based cloud infrastructures across a wide area network. The concept of a Flash Network Slice (FNS) has been proposed, a network resource that can be provisioned and reconfigured in a timeframe that is compatible with current provisioning of virtual machines in a data centre. As CloNe considers a multi-provider scenario, the architecture defines roles, responsibilities, interfaces and control functions for providers to cooperatively implement global virtual infrastructures, on-demand and paid for according to use.

An initial version of the CloNe architecture was reported in deliverable D-5.2 *Cloud Networking Architecture Description* [2]. That version guided the design of the complete system prototype later reported in D-5.3 *Description of Final Prototype* [3] as well as providing a context for research into the implementation of management and security functions that make up components of the architecture. The architecture reported in this document is a refinement of that initial version based on the experience gained in these prototyping exercises.

The main advancements of this version of the architecture over the initial version relate to the definition of the FNS, the layering of the architecture, re-factoring functions in the management framework, and extension of the functions covered by security aspects.

The FNS concept has been expanded to include more detail on how constraints and network functions can be included in its definition, and a better understanding of the division between generic and implementation specific aspects has been derived. This is the result of practical application of the concepts in multiple technologies for the complete system prototype.

The functions and interactions of the infrastructure service provider have been more cleanly separated out into four layers: service, inter provider, intra provider and resource layer. The service and inter provider layers were intermingled in the initial architecture. These are now recognised as two different conceptual layers. The former deals with abstract infrastructure models and service objectives exchanged within the user-provider service model and enacted through the infrastructure service interface. The latter deals with collaboration among peer providers and enacted through the Distributed Control Plane (DCP), including negotiation of shared configuration details, discovery of capabilities, and management and security control functions.

The management framework of the architecture has been re-factored, identifying resource and performance monitoring and resource control as independent functions from fault management and resource management. Also, the Distributed Knowledge Plane (DKP) has been removed as a functional block. The DKP provided distributed information access. This is now viewed as an artefact of a distributed implementation design choice and not an architectural function.

The security aspects have been expanded to accommodate the service layer delegation concept. Infrastructure service users and infrastructure service providers may independently delegate access rights to, and implementation of, virtual infrastructure respectively without informing one another. This forms chains of responsibility that need to be accommodated in authorisation. Moreover, an intrusion detection function has been added to the architecture.

Finally, this document includes examples of how dynamic applications operate within the CloNe architecture to dynamically and elastically accommodate demand for their services. The two primary scenarios for these applications are *Dynamic Enterprise*, dealing with on-demand provisioning of business systems across data centres interconnected by networks, and *Elastic Video Distribution*,

dealing with on-demand provisioning of applications at the edge of the network.

Contents

1 Introduction

Cloud computing is a novel service delivery model enabled by virtualisation that supports cost-efficient usage of computing and storage resources, featuring dynamic scaling and on-demand pay-per-use services. While the adoption of cloud computing services has become a reality, it is still hard to guarantee access to those resources over the existing Internet. Existing network services are not dynamically allocated and not integrated with Infrastructure as a Service (IaaS) clouds. The lack of protocols and interfaces to allow for that vision to materialize is a hindrance.

The vision of Cloud Networking (CloNe) is to fully realize the potential of networking in the cloud computing paradigm. More specifically, CloNe aims at providing cloud network services compliant to application requirements in a dynamic and automated way connecting customers to the cloud and connecting different parts of the infrastructure, i.e., geographically distributed data centres. Considering the current state of the technology, this is achievable by enabling the co-existence of legacy and new networks via virtualisation of resources and by fully integrating networking services with existing cloud computing services. CloNe embraces legacy networks by creating an abstraction layer where those network services can be encapsulated.

CloNe can dynamically expand usage of resources integrating performance and fault monitoring for dynamic resource allocation. The CloNe solution is able to connect private networks and infrastructures together across provider boundaries and technology boundaries. CloNe leverages virtualisation techniques not only for migration but for flexibility provisioning different kinds of application services. It embeds the concept of federation among different service providers, namely network and infrastructure providers.

In fact, network requirements have been so far neglected when IaaS services are deployed. However, as more business critical applications move to data centres, cloud customers will demand the integration of networking services with cloud services. The network services proposed by CloNe are provided through Flash Network Slices (FNSs), an abstract network resource that hides complexity and provides guaranteed connectivity to its user. The network services will be used in two different occasions. First, a service connecting the customer to the data centre and secondly, a service connecting infrastructures in different data centres. Scalable Adaptive Internet Solutions (SAIL) proposes the integration of those network services with existing IaaS services. This missing element is what CloNe brings to the cloud computing paradigm. How that integration happens is the subject of CloNe's architecture.

More specifically, the primary objective of building the CloNe architecture is to define roles, responsibilities, interfaces, and a reference model for deploying complex applications using FNSs over multiple, heterogeneous, multi-operator computing and storage clouds. The architecture work was initially presented in the deliverable *D-5.2 (D-D.1) Cloud network architecture description* [2]. During prototyping phase, the architecture was put to test and capabilities of the cloud networking architecture were demonstrated. This demonstration not only shows the networking capabilities in the data centre, but also shows dynamic network provisioning in Wide Area Network (WAN) connecting data centres. Through implementation the details of the architecture were specified and tested. Experimental research through prototyping played an important role fine-tuning the details of proposed solutions and making the concepts developed in the first version of the architecture a palpable reality. This document presents the final refined version of the architecture where experiences from prototyping are included.

The architecture presented in this document differs from the initial one in the following ways: the

understanding of network functions and constraints of a FNS and its modelling has been deepened; the separation of concerns presented by the new layered model where a service layer dealing with information and service objectives has been defined; the changes to the management architecture whereby two functions (Resource and Performance Monitoring, and Resource Control) were made independent to increase modularity and reduce overlapping responsibilities; the Distributed Knowledge Plane (DKP), part of the previous management architecture, has been removed since it is seen as an implementation artefact and not an architectural function; the security has been added of a delegation aware authorisation service, as well as an intrusion detection function.

The rest of this Chapter is organized as follows: Section 1.1 reminds the reader of the two previously defined scenarios of CloNe. Section 1.2 presents a categorized list of requirements derived from the scenarios and use-cases. Section 1.3 presents research items that are within and outside of the scope of this work. Section 1.4 is intended to put our work in the light of recent publications and industrial initiatives flourishing in the area of cloud network. Finally, Section 1.5 presents the outline for the rest of the document.

1.1 Scenarios and use-cases

Having CloNe vision in mind, we started by defining the scenarios and use-cases for cloud networking. The scenarios were the base for defining the main players and the technical requirements, which formed the foundation of the architecture. The two scenarios that were defined were the Dynamic Enterprise and the Elastic Video Distribution. Under each of these scenarios, a number of specific use-cases that shows the benefits of CloNe's vision were specified. These scenarios and use-cases have already been extensively described in the deliverable *D-2.1 (D-A.1) Description of project wide scenarios and use cases*[4] and they will be briefly included here for completeness.

Dynamic Enterprise. This scenario presents and depicts the provisioning of IT/IS solutions from the cloud network ecosystem to the enterprise market. It illustrates how the infrastructure can be reconfigured to adapt to mobile users during planned or spontaneous events. One of the objectives of the dynamic enterprise prototype is to show the establishment of an elastic FNS, connecting cloud computing and storage resources from multiple data centres over the WAN.

Cloud computing has demonstrated the ability to flexibly scale services to provide on-demand and pay-per-use IT/IS solutions. With FNS capabilities, cloud networking introduces dynamic flexible network provisioning into the equation. An enterprise will be able to dynamically adapt its IT/IS services to include new remote locations, added functionalities and new entities within its boundaries in a swift and effortless manner in accordance with the business requirements dynamics. This flexibility allows an enterprise to go beyond scaling IT/IS services for its core operations to provision and connect these services for short and long term projects both internal and external. The use-cases defined within this scenario were: media production, remote auditing, business goal management, and virtual desktop.

Elastic Video Distribution. This scenario shows the offering of video and similar services from a cloud network ecosystem to the retail market with enhanced Quality of Experience (QoE), provided by distributed computational resources within the network or close to the users. It shows how virtual infrastructure can be dynamically deployed over multiple infrastructure service provider (data centre or network) domains. For example, a large gathering of people creates a demand on the network that is higher than what the network infrastructure is dimensioned for, causing the user experience to deteriorate.

The scenario implies a framework of distributed resources in a cloud model, meaning that cloud resources are geographically scattered inside the operator network in a more fine-grained fashion than traditional centralized data centre clouds. A concrete example of what this could mean is the placement of cloud servers in operator network edge sites. In this scenario, content providers will

no longer need dedicated physical servers but will use on-demand virtual servers. This means that the deployment will allow multiple content providers to share the same distribution and computing infrastructure in cloud model. The use-cases defined within this scenario were: elastic live video distribution, distributed gaming, elastic video on-demand distribution, and video conferencing.

1.2 Requirements

The use-cases mentioned above and detailed in [4] have generated a set of technical requirements which are fulfilled by the current architecture. The requirements were grouped onto three categories, according to the area they relate to. The first group are the requirements related to networking, the second to management, and the third to security. The list herein presented is not meant to be an exhaustive list of all requirements; instead, it is meant to bring focus to relevant requirements that have direct implications on the presented architecture.

The network requirements are (from general to more specific ones):

- Ability to control and manage heterogeneous network technologies
- Ability to scale up or down the network resource
- Ability to swiftly add and remove remote sites to an existing network service
- Ability to allocate network resources through a common service interface that is technology independent
- Automate the connection of Virtual Machines (VMs) deployed across geographically distributed data centres through the network
- Provide end-to-end traffic isolation for different tenants
- To allow for the specification of multi-tier application topologies and to have them automatically deployed in data centres and wide area networks
- Ability to scale independently the different tiers of multi-tier applications and apply firewall properties between tiers or amongst nodes
- Ability to specify Quality of Service (QoS) parameters associated to the network service (e.g., capacity)
- Ability to specify the type of network service needed (e.g., broadcast capabilities)

The management requirements are:

- Ability to adapt resource management to the current state of the infrastructure
- Ability to dynamically relocate resources to always satisfy the user constraints
- Ability to support the allocation of infrastructure with specific requirements (or constraints)
- Ability to select high level goals as metrics to optimise service deployment
- Ability a user has to specify the elastic behaviour of an application
- Ability to scale a service up and down automatically as a result of service usage
- Ability to perform root cause analysis and localization for fault-handling
- Monitoring algorithms that operate on physical and virtual layers, collecting and modelling performance behaviour

The security related requirements are:

- The user must be able to request a security property (location, standard, isolation)
- The provider must be able to check authorisation for access control
- The provider must be able to authenticate identities

1.3 Scope and Focus

Work Package D (WPD) committed to the mission of providing a unified system for creation, control, management and optimization of virtual infrastructures across multiple providers. Virtual infrastructures are composed of distributed compute, storage and networking resources and the provisioning is on-demand and in an elastic manner. The scope and focus of contributions of our work include:

- Proposal of new network service interfaces that can easily be integrated with existing IaaS services
- Protocols and mechanisms allowing a distributed virtual infrastructure to be deployed across data centres and wide area networks
- Proposal of enhanced IaaS interfaces where advanced networking features are included
- Definition of customer facing interfaces allowing for deployment of distributed virtual infrastructures and specification of application requirements and goals
- Basic authentication solutions for cloud networking where delegation of infrastructure may happen
- New management mechanisms to optimize the use of resources in a data centre

It is not in the scope of architectural contributions:

- Defining new network virtualisation techniques (e.g., VXLAN, OpenFlow)
- Defining new routing or switching solutions to WAN or data centres (e.g., Multiprotocol Label Switching (MPLS), Virtual Private LAN Service (VPLS))
- Defining new path computation engines for existing networks

The CloNe architecture completes the cloud computing picture by addressing networking aspects introducing the concept of FNS, which is a network resource that can be provisioned and dimensioned on a time scale comparable to existing compute and storage resources. Through FNSs, CloNe also provides dynamic connectivity services on-demand mirroring the pay-as-you-go business model of existing cloud services.

1.4 Recent related work

In the past years, the area of cloud networking has started to gain more attention in the academia and industry. At the beginning of the SAIL project very little work in the area existed. Today, the term is established and the amount of papers published at academic conferences has been steadily increasing. A proof point is the number of papers produced by the foremost networking conference (SIGCOMM 2012 [5]) on data centre related issues, which was of 9/31 (29 percent).

Given the attention and traction around this research area, a new IEEE international conference has been created (1st IEEE International Conference on Cloud Networking 2012). CloNe partners have had two papers accepted in that conference, one being nominated for Best Paper Award. SAIL

has organized in conjunction with the EU project GEYSERS [6] a workshop on Cloud Networking at FuNeMS [7].

A recent work relevant to part of CloNe's work was presented at SIGCOMM 2012 [8]. The authors of that paper present a network aware service placement method that moves a service in a distributed data centre environment depending on the network conditions and the user location. CloNe however considers the application requirements provided in the form of constraints or goals. Those are then mapped to an IaaS provider and continuously optimized if network conditions change.

Aiming at sustainability, [9] introduces electricity cost and carbon footprint as another variable along with server workload when redirecting a user request to an appropriate data centre. CloNe provides energy efficient data centres through the creation of a specific management objectives (i.e., server consolidation) that can be used in a data centre [10]. CloNe has designed and developed two data centre management methods for VM placement, one tailored at the initial placement and the other at live optimizations, both featuring the choice of different management objectives.

Not only academia, but also various standardization bodies have started efforts in the area of cloud computing (e.g., OGF), cloud networking (IETF's Network virtualisation Overlay Working Group), cloud management (DMTF CLOUD Working Group), amongst others. Even though the number of standardization bodies focusing on cloud computing has increased, the scope and responsibility of each one of them is sometimes overlapping and/or limited. This unfortunately creates a scattered standardization picture that will most likely improve as the technology matures.

It is important to emphasize the fact that most of the *de facto* standardization work in cloud computing and cloud networking has been driven by the open source community. Citrixes' CloudStack was released as open source under the Apache Software License. The most prominent open source cloud management system is OpenStack that in a very short time managed to create a large community of contributors and strong industry support. Support for advanced networking is slowly progressing in OpenStack. As of October 2012, the latest development is the acceptance of Quantum [11], a layer 3 network service for data centres, as a core project in OpenStack. CloNe has released parts of its work as open source: libnetvirt, pyOCNI, and Cloud Message Brokering Service (CMBS).

1.5 Document Outline

This document is organized in a way to provide coherent and complete overall view of CloNe architecture. Chapter 2 presents the main components of the architecture, interfaces and protocols. An introduction to the management and security aspects that will be further refined later in the document is found there. Chapter 3 then presents the network model (FNS) and mapping to specific network technologies. Chapter 4 presents the solutions to the management of resources within one provider, while Chapter 5 focuses on inter-provider questions. Chapter 6 presents the envisioned way a customer/tenant will interact with the CloNe system and how the internal components are put together to implement the proposed use-cases. Finally, Chapter 7 presents closing remarks.

2 The CloNe Architecture

This chapter describes the CloNe architecture at a conceptual level. Throughout we make reference to four roles that actors may adopt in any given context: tenant, infrastructure service user, infrastructure service provider, and administrator. So we begin by defining these roles as follows:

- **Tenant** participates in a business relationship with an infrastructure service provider in which the tenant agrees to pay for the provision of virtual infrastructure.
- **Infrastructure Service User** accesses a virtual infrastructure service in order to obtain, examine, modify and destroy resources owned by a tenant.
- **Infrastructure Service Provider** offers an infrastructure service to tenants that may be accessed by infrastructure service users to obtain, examine, modify and destroy resources.
- **Administrator** has administrative authority over underlying virtual or physical equipment (the administrative domain) used to implement virtual resources.

2.1 Overview

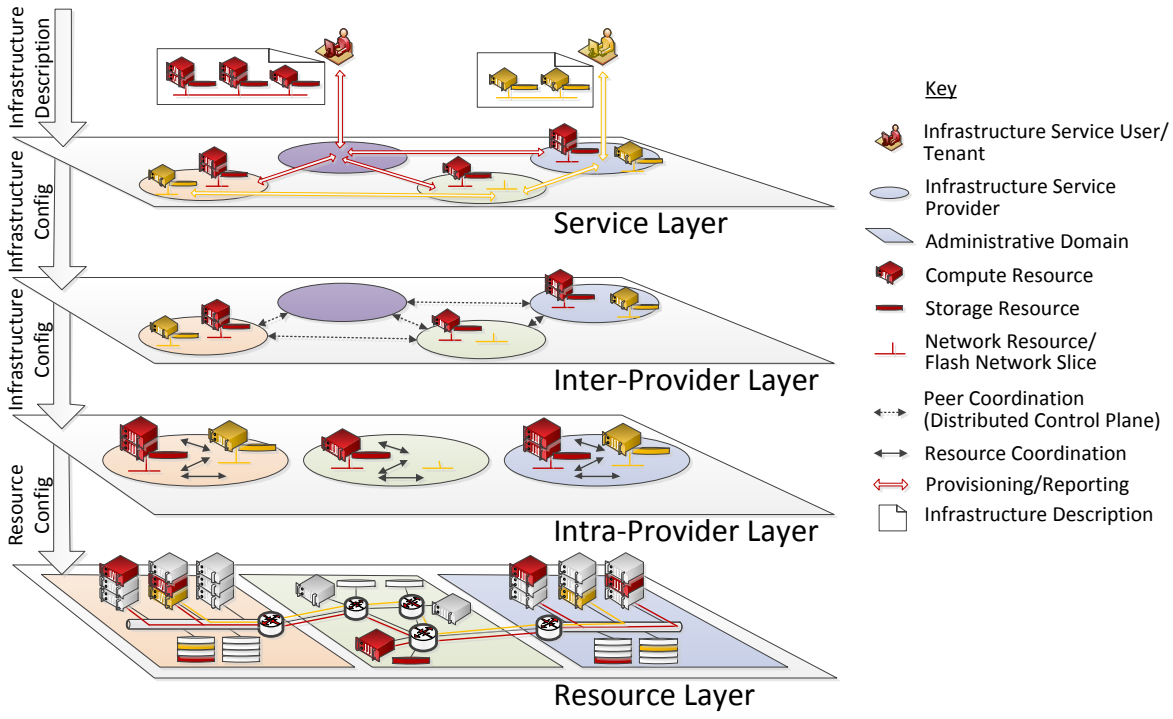


Figure 2.1: Architecture overview

The architecture is organised according to the four layers depicted in Figure 2.1. The resource layer is concerned with virtualisation of underlying equipment to implement individual virtual resources. The intra-provider layer deals with organisation and coordination of virtual infrastructures within a single infrastructure service provider. The inter provider layer is concerned with coordination that has to occur between providers where they collaborate to interconnect virtual infrastructures. The service layer renders the virtual infrastructure service itself and accommodates business relationships among the actors.

The colours of the infrastructure service providers and administrative domains represent the same actor operating in each layer. As can be seen from Figure 2.1 not all infrastructure service providers have an administrative domain or appear in the lower layers. The inter-provider and service layers may contain providers whose sole purpose is to coordinate virtual infrastructures implemented by other providers and to cooperate in presenting them as a service for tenants.

Non-collaborative infrastructure service providers, such as those that exist today, also fit in this architecture, but would be absent from the inter provider layer. Such providers operate in isolation and do provide service, but do not interact with their peers to coordinate their services. This case is not shown.

CloNe introduces a new network resource called the FNS. This is implemented by network virtualisation technologies in the resource layer and represented as a virtual resource within virtual infrastructures in the other layers. CloNe defines a new virtual infrastructure information model to include the FNS. The infrastructure service users interact with the infrastructure service providers by exchanging descriptions based on this information model.

CloNe also extends the infrastructure service functions to enable collaboration among service providers. This service model is implemented by the service layer.

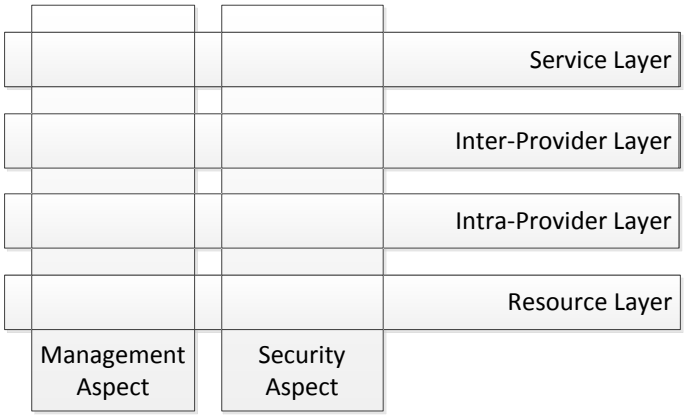


Figure 2.2: Management and security aspects occur in each layer of the architecture

Management and security aspects cut across all layers of the architecture as shown in Figure 2.2. Each intersection between management or security aspects and an architecture layer represents functions implemented in different interfaces or with a different scope of control. The functions come together within each aspect to provide coherent operation of the service.

The extensions to the service model, the infrastructure description (including the new FNS network resource and information model), the architecture layers, and the management and security aspects are described in the remainder of this chapter. The subsequent chapters describe concrete implementations developed in prototypes of the architecture.

2.2 Service Model

Infrastructure as a Service (IaaS) is a business relationship between the tenant and infrastructure service provider roles in which the tenant pays the infrastructure service provider to implement virtual infrastructure. This relationship is depicted in Figure 2.3 in which a customer organisation (acting as tenant) obtains access to virtual infrastructure implemented by a cloud operator (acting as infrastructure service provider).

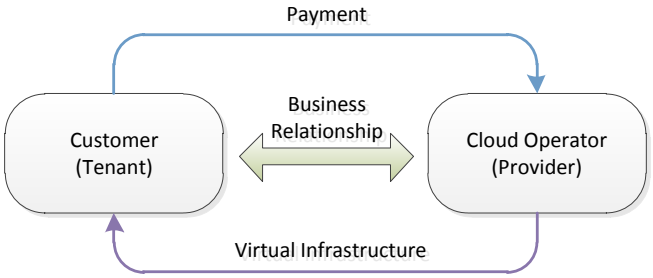


Figure 2.3: The IaaS service model

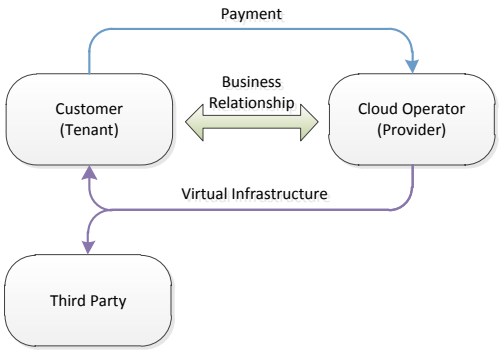
This simple arrangement is already implemented by existing cloud operators. CloNe extends the IaaS functions to include collaboration among infrastructure service providers by adding mechanisms to support *delegation* and *cooperation*.

Delegation

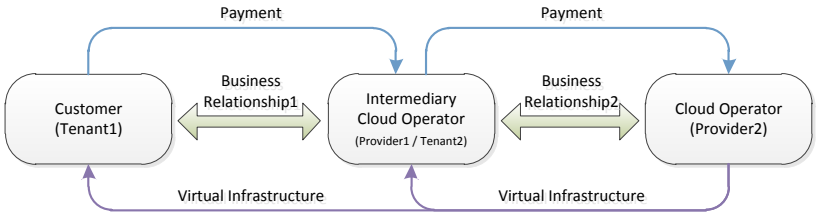
The CloNe infrastructure service uses delegation to allow additional entities to participate in the business relationship. There are two forms of delegation, as depicted in Figure 2.4.

The tenant is able to delegate the right to limited access or control of its virtual infrastructure to a third party, including the right to create new infrastructure. This changes the nature of the relationship as depicted in Figure 2.4(a). As shown, the third party also obtains access to the virtual infrastructure from the provider, but does not pay for it. The third party is viewed as representing, or acting on behalf of, the tenant and it is the tenant that retains ownership.

An infrastructure service provider is able to exploit a similar behaviour to delegate the implementation of virtual infrastructure to yet another provider. As shown in Figure 2.4(b), this involves two business relationships: one between the customer organisation (the first tenant) and the intermediary cloud operator (acting as a provider); and a second between the intermediary cloud operator (this time acting as a tenant) and another cloud operator (the second provider). Instead of implementing the virtual infrastructure directly, the intermediary contracts the other cloud operator to implement it under a second business relationship between them: this infrastructure is owned and paid for by the intermediary. The intermediary can delegate access rights to the customer organisation and can mediate management interactions, giving the impression that the virtual infrastructure is provided by the intermediary. In return it charges the customer, thus fulfilling its contract under business relationship1.



(a) Controlled delegation of access rights



(b) Controlled delegation of implementation

Figure 2.4: Delegation in the CloNe IaaS service model

Cooperation

The CloNe infrastructure service includes the notion of cooperation across cloud operators, as depicted in Figure 2.5.

A customer organisation may be the tenant of multiple cloud operators and may wish to interconnect virtual infrastructures implemented by each. Each cloud operator will be in a different business relationship with the customer and will view the customer as a different tenant. The cloud operators may need to interact directly with one another to coordinate the interconnection. The customer can enable this interaction by delegating appropriate access rights to each cloud operator. The operators may now charge the customer (their respective tenants) for the virtual infrastructure they provide and for the interconnection that has been authorised.

This model of coordination naturally applies to the case of two different customer organisations choosing to interconnect their virtual infrastructures.

Business Models

There are a variety of business models in use for existing cloud infrastructure services. Pricing mechanisms range from spot markets for virtual machines, to escalating pricing for network bandwidth, to fixed pricing for various units of resource. Entire cloud infrastructure service implementations can be contracted as a private managed service. CloNe does not seek to impose a specific business model, nor does it attempt to anticipate future business arrangements.

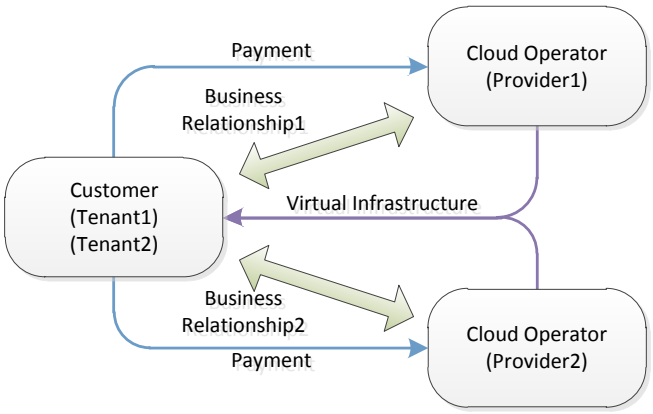


Figure 2.5: Coordination in the CloNe IaaS service model

The delegation principle provides complete flexibility for providers to come to business arrangements among themselves, without exposing those arrangements to customers or limiting the possible business models available to customers.

2.3 Infrastructure Description

Most components and interfaces of the CloNe architecture need to communicate descriptions of virtual infrastructure. Functions in the resource layer receive detailed resource configuration requests; the intra-provider layer receives configurations describing groups of resources and their relationships; the intra-provider layer exchanges information among providers; and the service layer responds to high level requests for virtual infrastructure and reports on its status. CloNe introduces an information model and data description techniques that facilitate these interactions throughout the architecture.

The CloNe virtual infrastructure includes a new network resource type, called the FNS, that supports the service model described above. The following introduces the FNS and an information model that includes it. In addition, data description techniques that support the delegation and cooperation aspects of the service model are described.

2.3.1 Flash Network Slice

The *Flash Network Slice (FNS)* is a concept of a virtual network introduced by CloNe with the following features:

- It is a network resource providing a message forwarding communication capability.
- It can be connected to VMs and other FNSs
- It can be connected to FNSs across administrative domain boundaries.
- It has quality of service properties and network functions associated with message forwarding between resources connected to it.

- It is established automatically and in a time frame comparable with existing virtual infrastructure resources such as VMs.

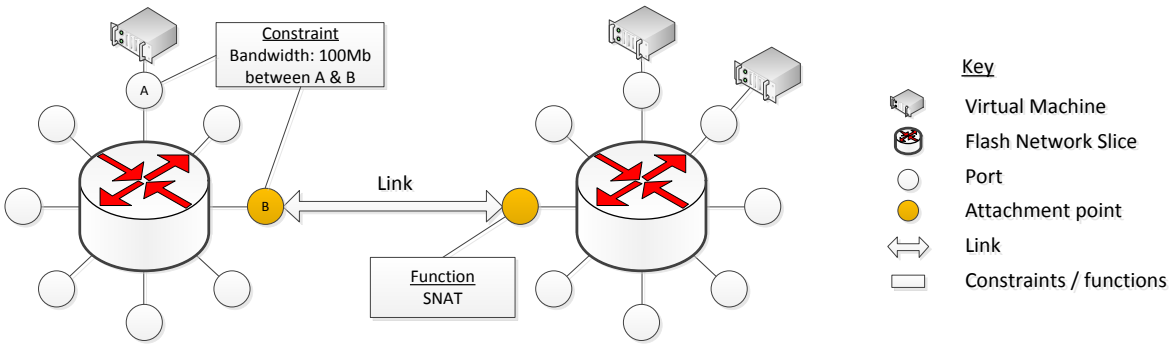


Figure 2.6: Virtual infrastructure spanning two administrative domains

The FNS, depicted in Figure 2.6, is based on the typical cloud computing concept of a virtual network as a resource that can be provisioned within an administrative domain. It has a number of ports that are access points to its message forwarding service. VMs can be connected to a port via network interfaces. Two FNS can be linked together between special ports, called attachment points, that extend the message forwarding behaviour across the link. FNSs linked in this way implement a larger virtual network that is functionally equivalent to a single FNS. This feature can be used to construct a network from portions that are under the control of different administrative domains or that have different implementations. The interconnection of FNS can also be used to connect two independent infrastructures.

The FNS is a generic concept that does not specify what type of communication occurs beyond message forwarding. It can be specialised to specific types of network service or to specific network technologies such as a layer 2 network that transports Ethernet frames or a layer 3 network that transports IP packets. In the former example, the ports would be assigned MAC addresses, the FNS would correspond to a broadcast domain and two such FNS could be linked by a layer 2 tunnel between tunnelling attachment points. In the later, the endpoints would additionally be assigned IP addresses, but the broadcast domain need not be extended across a link between FNS.

Ports can have properties and functions associated with them. A property defines a constraint on the port or between ports. A function defines a network function applied at the port. Multiple ports may have access to the same network function. This can be used, for example, to apply filtering or address translation at attachment points, or to provide a network service to all ports on an FNS.

The mapping of FNS concepts to implementation technologies is described in detail in Chapter 3.

2.3.2 Information Model

Figure 2.7 is a UML class diagram representing a virtual infrastructure at the conceptual level. This representation is highly influenced by existing cloud computing information models. The following describes the main classes of the model.

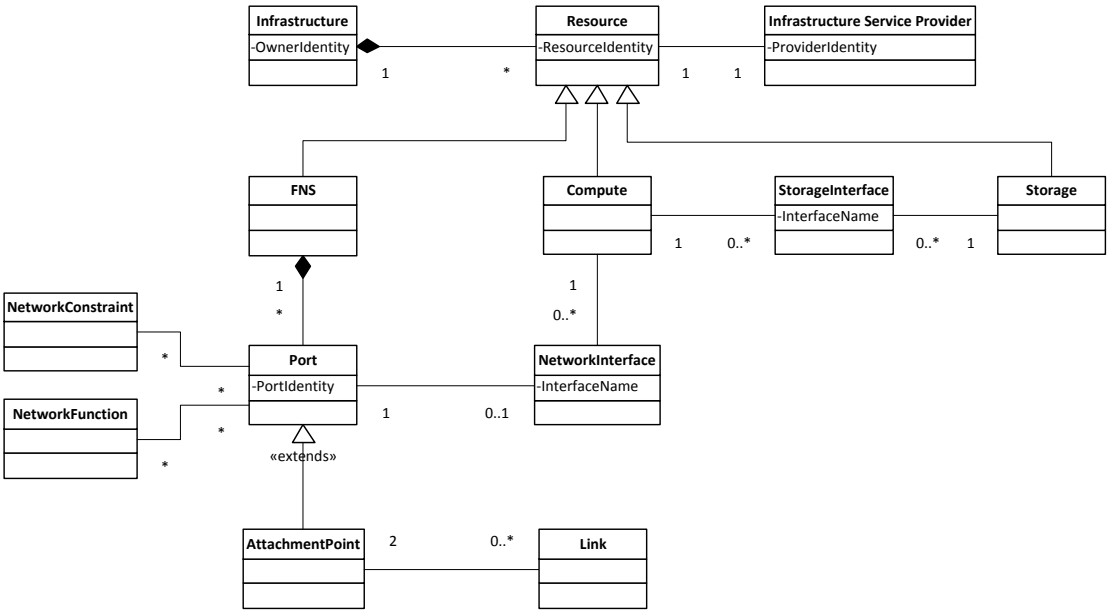


Figure 2.7: User view of virtual infrastructure

Resources

The resource is the basic class in the information model corresponding to the virtual resource concept. The three types of resource are *compute*, *storage* and *FNS*. Compute represents a virtual compute resource, such as a virtual machine. Storage represents a virtual storage resource, such as a volume. An FNS is the additional CloNe resource type that replaces the usual network resource providing on-demand guaranteed end-to-end message delivery.

Flash Network Slice

The FNS class and the associated Port, AttachmentPoint, Link, NetworkConstraint and NetworkFunction classes represent the FNS concepts described above. The FNS has a (possibly empty) collection of Ports that can be associated with the NetworkInterface class used to connect Compute information objects. The AttachmentPoint is the specialised version of a Port used with Links to represent interconnection of FNS.

The FNS, Port and Link form an abstract representation of a virtual network that can be specialised to define implementation details as described above. The classes defined here represent implementation independent characteristics that apply to all FNS regardless of implementation.

Port

A Port may have NetworkConstraint and NetworkFunction classes associated with it. NetworkConstraints represent characteristics of the port, such as bandwidth or latency, and may be associated

with multiple ports. For example, a NetworkConstraint may represent a minimum bandwidth for a particular port, between two specific ports, or between all the ports of an FNS.

A NetworkFunction represents an *in-network* function applied to the messages sent to or from a port. Using NetworkFunctions in combination with Links between FNS it is possible to represent a restricted communication flow between FNS or to specify a boundary between different addressing schemes in the two.

An AttachmentPoint is a special type of port in that it extends the message forwarding behaviour of the FNS across a Link. Ultimately, it is expected that an FNS is implemented within the control of a single autonomous administrative domain, whereas a Link may be coordinated across the boundaries of administrative domains. In this case the AttachmentPoint represents the endpoint for communication between domains.

Infrastructure

The Infrastructure class corresponds to the virtual infrastructure and represents a collection of resources owned by a single tenant (OwnerIdentity) and related to a single infrastructure service provider (ProviderIdentity). A resource can be part of only one infrastructure, but a single tenant may own multiple infrastructures at a single provider, implying multiple separate collections. The FNS of two infrastructures can be interconnected by Links.

This definition of an infrastructure implies that if a tenant owns resources at two different providers they are necessarily part of two different infrastructures. This is indicative of the fact that the tenant has independent relationships with the two providers, as described in Section 2.2. The case of an infrastructure service provider delegating implementation of an infrastructure to another provider is not represented in the model. In practice the delegating provider (called the intermediary in Section 2.2) would generate a new model of those infrastructures, representing a transformation of the original, and would be responsible for maintaining the relationship between the models.

2.3.3 Data Description and Interchange

The information model can be given concrete specification by encoding it using a data description language. Two variations of data description have been developed in CloNe, including Virtual private eXecution infrastructure Description Language (VXDL) [12], a resource description language used to describe infrastructure topologies including networks, and Open Cloud Networking Interface (OCNI) [13], an extension the Open Cloud Computing Interface (OCCI) standard [14, 15] currently used as the interface to existing cloud computing infrastructure managers such as OpenNebula [16] and OpenStack [17]. These can be exchanged in interchange formats as documents and have been used in the prototype described in [3].

A useful feature of data descriptions is the *reference*, a link to information defined elsewhere.

References

References are implemented by Universal Resource Identifiers (URIs) used to locate data descriptions that may be obtained on demand. References allow information models to be partitioned into multiple data description documents that can be communicated and retained separately. This separation can be used to isolate information for access control purposes: references can be passed around, but only an authorised entity can resolve them, contributing to our information hiding objective (i.e., providers may not be willing to disclose information to each other). Similarly, they allow late binding of information, supporting the transparent transformation and indirection required for delegation.

A single provider does not need to be aware of all the infrastructure owned by a particular tenant, it only needs to know its own part. This fact can be used to partition a virtual infrastructure into separate data descriptions, each restricted to the resources that are under the control of the provider receiving the description. Where a description must identify resources that are under the control of another provider, they can do so using a reference. For example, a Link between AttachmentPoints may be shared by two infrastructure service providers, whereas each AttachmentPoint is implemented by only one provider. Each provider receives a copy of the Link object in a data description document, but only its own AttachmentPoint is described. The Link identifies the remote AttachmentPoint using a reference. If the provider needs to obtain details about the remote AttachmentPoint it can resolve the reference. The remote provider is able to control access to the information.

The resolution process itself is described in Section 2.4.3.1. In CloNe we use references based on the resource naming schemes of the provider interfaces, such as OCCI/OCNI. It is possible to use different naming schemes for data objects accessed through different interfaces (identified by the initial locator part of the URI). The Network of Information (NetInf) naming scheme described in [18] is an alternative URI based scheme that can be used for data objects accessed through the NetInf name resolution interface.

2.4 Architecture Layers

The layers of the architecture build on each others functionality from the bottom up. They are described in that order in the following.

2.4.1 Resource Layer

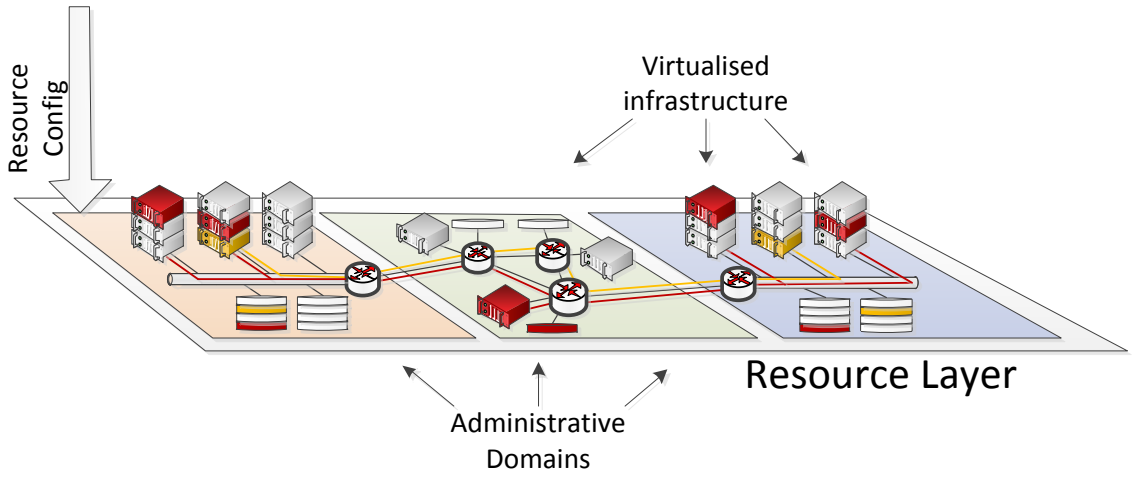


Figure 2.8: The Resource Layer

The primary role operating in the resource layer is the administrator. The layer deals with individual control of virtual resources within a single administrative domain. The virtual resources can be individually identified, assigned certain properties, and have runtime status. They may also have links to other virtual resources that are managed separately, for example, a virtual machine might be linked to a virtual network and a storage volume.

The correct operation of a virtual resource may depend on the status of resources it is linked to; however, its management subsystem may not have visibility of the linked resource if it is managed by a different subsystem. This level of coordination is outside the scope of the resource layer.

2.4.1.1 Resource Administration

The resource administration interfaces correspond to the management functions that are used by the *administrator* role to create, manage and destroy virtual resources within an administrative domain. In general these interfaces are the management interfaces of some virtualisation technology such as the libvirt [19] virtual machine manager interface, a storage device controller or a network administration interface.

The resource administration interfaces are implementation specific. They must provide information about the underlying infrastructure including the network topology and technologies used, so that the administrator can make a decision on how to manage the resources and what information needs to be passed through these interfaces. Each interface (compute, storage or network) therefore, takes specific configuration details from an administrator in order to configure resources according to the infrastructure service user's needs.

The following are examples of parameters and functions that could be present in these interfaces.

Compute Resource Interface

This interface provides technical capabilities similar to well-known interfaces like libvirt. At the same time it could be augmented with the ability to provide more advanced capabilities than pure virtual machine control (e.g. load balancing through virtual machine migration and distribution of compute tasks onto various machines). This interface provides access and ability to invoke a number of essential functions in handling resources such as:

- Creation/Deletion/Start/Suspend/Stop of VMs
- Compute service query and configuration to set performance target and express desired compute service characteristics
- Selection of software such as OS and execution environment

Storage Resource Interface

The storage service interface can very much rely on standards such as Cloud Data Management Interface (CDMI) and possibly on the de facto standard Amazon Simple Storage Service (S3) to provide access to virtual storage spaces as well as physical storage. Compatibility with such standard is essential as they are widely adopted in the cloud community. Additional needs emerge when network providers also provide storage within the network nodes such as caching and even storage of files, documents or multimedia files or data.

Network Resource Interface

Compute and Storage can be allocated and managed as cloud resources via well-defined web interfaces and Application Programming Interfaces (APIs) such as those mentioned above. These kinds of interfaces and APIs are cloud computing and storage specific. What is missing today are the cloud networking interfaces and APIs that CloNe intends to add or introduce.

The objective is to define these missing interfaces and APIs so cloud networking can be achieved like traditional network configuration for Network Interface Cards (NICs), Virtual Local Area Networks (VLANs), OpenFlow, OpenvSwitch, Dynamic Virtual Private Networks (VPNs). This

interface will highly depend on the capabilities of the underlying network and existing network management systems. It is expected that one should be able to configure parameters like bandwidth and jitter. Mobility should be supported, as well as fault monitoring, and redundancy, depending on the capabilities of the underlying network. One could also set triggers for monitoring of SLA parameters.

The availability of specialised cloud networking interfaces and APIs will facilitate deployment, configuration, management and supervision of networks as an integrated part of private and hybrid cloud establishments.

2.4.2 Intra-Provider Layer

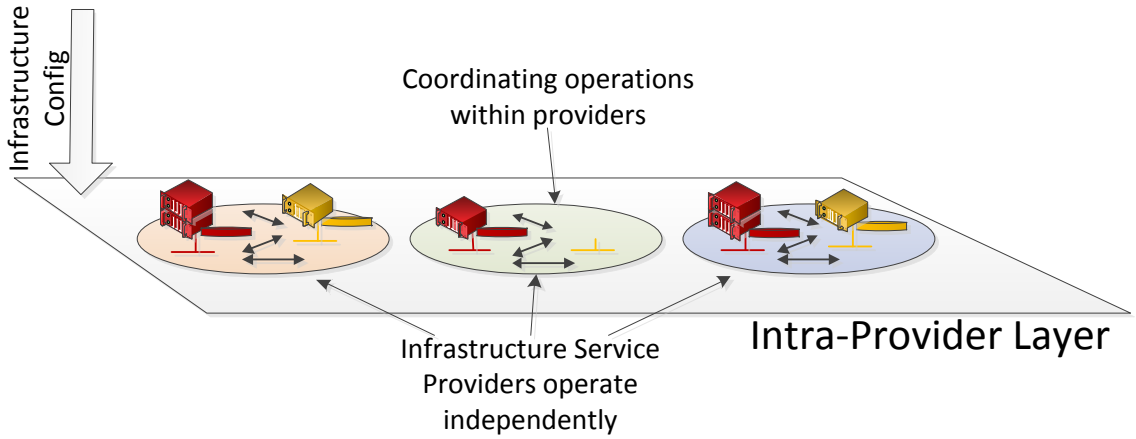


Figure 2.9: The Intra-Provider Layer

The primary role in the intra-provider layer is the infrastructure service provider, which also directs actions carried out in the administrator role at the resource layer. The intra-provider layer deals with collective control of multiple virtual resources within a single administrative domain. The links among these virtual resources determine the topology of the infrastructure and constrain their combined management behaviour.

An intra-provider infrastructure is managed by a single administrative authority that has management rights over the underlying equipment and virtualisation technology. As a consequence, within an intra-provider the administrative authority has full knowledge about the available virtual resources and virtualisation capabilities at any time and can use that knowledge to manage all the resources of a virtual infrastructure as a collection.

At this layer the mapping between the virtual infrastructure and the underlying equipment can be determined. This mapping can take into account group allocation (all or nothing allocation of a collection of virtual resources) and optimal placement (relative placement of virtual resources or use of underlying infrastructure). For example a VM could be placed in a location with optimal network performance relative to a given end user.

Some technology selections can be made at this layer. A virtual machine could be executed on a choice of different servers with different memory sizes or chip sets giving different performance trade-offs; a disk volume could be placed on local storage or network attached storage; a network link could be mapped to an isolated VPN tunnel or an open shared network.

Optimal placement and technology selections will depend for the most part on private policies of

the administrative authority for the domain. However, the properties of the virtual resources and their links and the properties of the virtual infrastructure as a collection will influence the choices, in some cases dictating minimal requirements for the virtual infrastructure.

The management and security functions that operate in the intra-provider layer have been studied extensively in CloNe, with many examples reported in Chapter 4. Almost all the functions described for management in Section 2.5 and for security in Section 2.6 exist in this layer.

2.4.3 Inter-Provider Layer

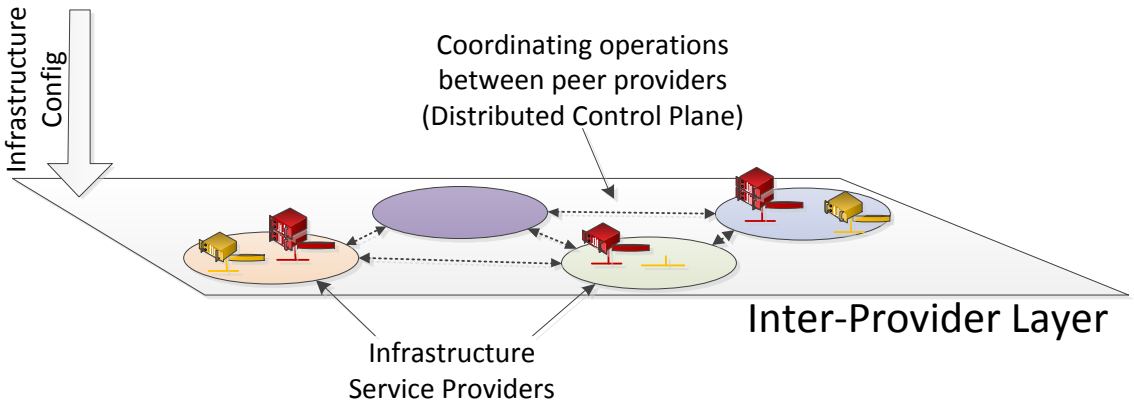


Figure 2.10: The Inter-Provider Layer

The primary role in the inter-provider layer is the infrastructure service provider, with multiple providers interacting across their administrative boundaries as peer groups. The layer deals with collective control of multiple virtual resources across multiple administrative domains. An inter-provider infrastructure is the composition of multiple intra-provider infrastructures. An intra-provider infrastructure may contain virtual resources that have links with virtual resources in other intra-provider infrastructures, thus connecting the virtual infrastructures and determining the topology of the inter-provider infrastructure.

The inter-provider layer shares information and performs coordination across the multiple providers, which act autonomously at the intra-provider layer. The state of underlying equipment and virtualisation capabilities are likely not fully shared beyond domain boundaries, but providers may exchange limited information that they are willing to expose about their domain in order to facilitate resource optimisation decisions and to coordinate inter-provider links.

2.4.3.1 Distributed Control Plane

The Distributed Control Plane (DCP) describes a category of protocols, interfaces and control operations within the inter-provider layer that enable two or more *infrastructure service providers* to interact and exchange information. The following lists a minimal set of necessary interactions that are part of the DCP. A more precise definition of the protocols and interfaces that constitute the DCP is given in [3].

- **Capability discovery**: the process by which providers advertise their existence and their capabilities with their peers.

- **Reference resolution**: the process of converting an abstract representation of remote information (i.e. data model references described above in 2.3.3) to the actual information. As an example, if a network link is to be established, information about the remote attachment point for the link may be required. That information may be represented by a reference that can be resolved through a resolution protocol to obtain the actual information. Reference resolution is described in more detail in Section 5.1.3.3.
- **Link negotiation**: the process of two providers negotiating implementation specific configuration details to establish a cross provider link between attachment points. A link negotiation protocol that can handle multiple network technologies is detailed in [3] and described in Section 5.2.2.
- **Authorisation**: the process of determining the right to access and manage virtual infrastructure. This is complicated by the chain created when users delegate access rights and providers delegate implementation. A distributed authorisation mechanism is detailed in [3] and described in Section 5.1.4.1.

The DCP is generally concerned with distributed coordination and global information access. Communication between providers on DCP does not need to be synchronous. The specific protocols and interfaces used may depend on the specific relationship between domains and technology used. However, generic protocols may be employed to implement common coordination and communication services.

2.4.4 Service Layer

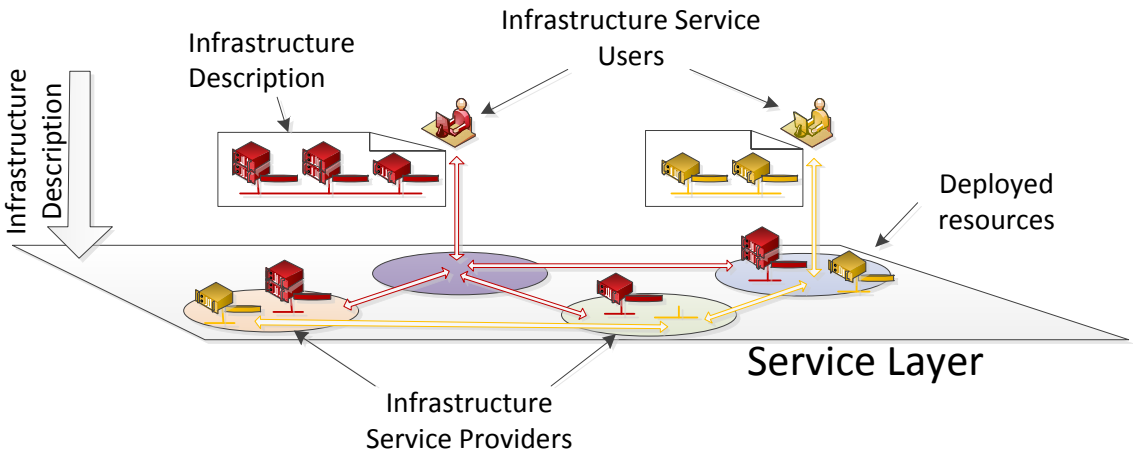


Figure 2.11: The Service Layer

The main roles operating in the service layer are the infrastructure service user and the infrastructure service provider. The tenant also has an important role in this layer as the root of authorisation for an infrastructure service user to act.

The CloNe infrastructure service is implemented by a set of interfaces and functions that enable the creation, monitoring and management of virtual infrastructures, including automatic delegation and collaboration across providers.

2.4.4.1 Tenant registration

Tenant registration refers to the process of a tenant establishing a business relationship with an infrastructure service provider. This may be an automatic process, in which a tenant discovers a provider's capabilities and programmatically registers as a customer. Equally, it may be a manual process. For example, a network operator and a data centre operator may enter into a mutual business relationship, by which the network operator installs a physical network connection to a data centre operator. One or both may register the other as a tenant of their services.

2.4.4.2 Infrastructure Service

The infrastructure service interface is implemented by each infrastructure service provider. Users interact with a single provider to deploy and manage a virtual infrastructure under the authorisation of a tenant of that provider.

Infrastructure service requests are performed using a high-level description language based on the information model described above. Requests for virtual infrastructure need to be decomposed into parts for delegation across multiple providers where appropriate. The decisions to delegate are based on information about the capabilities of other providers at a level they are willing to share, and detailed knowledge of the local provider capabilities, policies and business relationships. If the provider decides to delegate infrastructure it does so acting in the role of an infrastructure service user, generating a new request that it issues to another provider via its infrastructure service interface. The provider is responsible for maintaining a mapping between the original request and the new one delegated to the other provider, mediating further requests related to the delegated resources through the infrastructure service interface, interpreting information reported back for the original user to consume, and issuing appropriate access authorisation to allow the original user to interact with the resulting virtual infrastructure.

Information that supports the decisions taken in the service layer originates from other layers. Other providers share capability information through the inter-provider layer, local capabilities are known through the intra-provider and resource layers. The infrastructure service user's request contains constraints that guide the decision process.

The objective is to allow the user to specify high-level goals in the form of system service level agreements that will be automatically broken down into low level control actions. This language should define service goals that cross the boundaries between different cloud providers. It should also address functional and non-functional requirements of the users, which may be constraints that end-user business processes may impose for example on process migration.

The goal translation performed at the service layer is performed in concert with management functions at other layers as described in the management aspects in Section 2.5 and practical examples are given in Chapter 4.

The infrastructure service requires the use of well-defined security modules, in order to satisfy its security requirements, provided through user, operator and legal requirements, i.e. authentication, auditing, confidentiality, integrity and assurance, besides others. Security goal translation handles the realisation of security requirements on the underlying resources, with the help of external modules, for example an auditing module and an access control policy module. This shall be covered in the Section 2.6.

2.5 Management Aspects

The aim for the management architecture is to allow overall optimization of service provisioning and maintenance by considering network and computing resources as a unified whole, operating across a distributed resource pool using a common management framework. The management concept

allows the infrastructure service user to request services on the basis of its current and recognized infrastructure needs. The management concept enables configuration and deployment of optimal and efficient solutions, as well as dynamic reconfiguration and compensation for deviations from the agreed QoS-levels of requested services or for elastic variations of consumer demands during the service lifetime.

This aim leads to several challenges concerning scalability, efficiency, and adaptability of the management functions under increasing network complexity. The challenges lie in developing concepts and algorithms that can meet the technical, legal, and commercial requirements and ramifications of CloNe. The five main challenges are listed below:

- The business model of CloNe allows rapid deployment of services. This leads to challenges in developing efficient and fast algorithms for, e.g., optimal resource selection, performance modelling, and service migration.
- Multi-tenancy leads to resource sharing which implies challenges for SLA compliance as resources may suffer from starvation and result in performance degradation.
- Multi-tenancy together with delegation and volatile service provisioning (migration, multi-provider) implies challenges for fault localization (root-cause analysis). This further affects and challenges the optimization aspect of goal translation.
- Equipment heterogeneity and possible restrictions of access to underlying network information lead to challenges for capability and resource discovery.
- The (geographically) distributed nature of infrastructure services leads to challenges in resource configuration due to possible jurisdictional differences (with respect to encryption and data restrictions, for example) between parts of the same service.

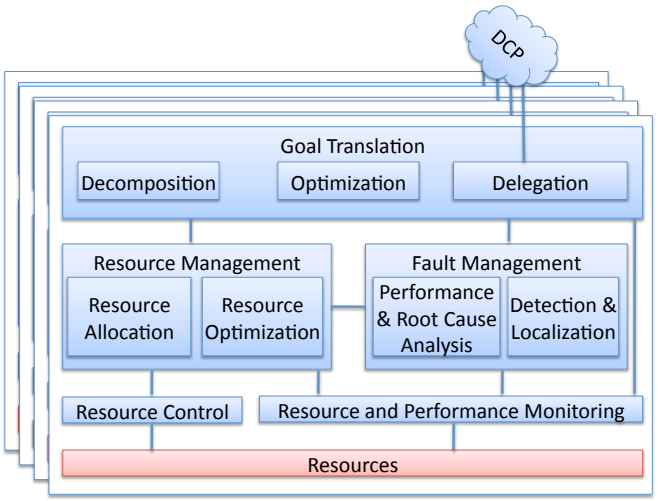


Figure 2.12: Management Concepts

2.5.1 Management Architecture

The management architecture builds on five management concepts: *goal translation* (for mapping requests to optimal virtual infrastructure provisions), *resource management* (for optimally mapping virtual infrastructure to underlying equipment), *fault management* (for maintaining performance levels through modelling and fault mitigation), *resource and performance monitoring*, and *resource control*. Figure 2.12 gives a schematic view of the management concepts. The concepts are implemented across the architectural layers by corresponding management functions. The management functions together provide the necessary means for service request management, through communication and exchange between the functions. The functions and their interaction are described in Chapters 4 and 5.

Goal Translation Concept

Goal translation is implemented both in the service and the inter-provider layers providing layer-specific functionalities. Goal translation interfaces with the intra-provider layer for obtaining provider capabilities, and with the resource layer for obtaining performance and resource monitoring data and models, which are used in the process of selecting optimal infrastructure provisions. For its operation in the intra-provider case, goal translation relies further on communication with fault and resource management functions, while in the inter-provider case it communicates with corresponding inter-provider functions.

The concept of goal translation facilitates the use of a high-level service specification language, in which an infrastructure service user can request services according to its recognized needs (which, could be merely in terms of system or service level constraints on the required performance). Such high-level requests are translated into lower level goals. In the inter-provider case, the translation amounts to supporting delegation and decomposition into goals that can be optimally matched by the single provider components based on their capabilities. In the intra-provider case, the translation amounts to selecting optimal infrastructure solutions among candidate solutions provided by resource management functionalities. In this case, the translation is based on performance models provided by performance monitoring and modelling management functions. In the general case, both for intra- and inter-providers, the translation may have to take several constraints into account simultaneously (including constraints originating from the business model of the provider as well as from technologic or environmental considerations). Thus, optimality need not be uniquely defined and may rather best be defined in terms of an equilibrium (e.g., Pareto optimality). The performance of the selected solution is monitored with respect to the low-level goals (performance objectives or subgoals). A failure to meet a low-level goal may lead to re-optimization of the corresponding high-level goal in order to find alternative provisioning solutions.

The goal translation concept thus contributes to the robustness and resilience of service provisioning by allowing uncertainty to be encoded in the service request and by isolating the tenant from the actual implementation choices of the service.

Fault Management Concept

Fault management consists of a set of algorithms that operates in collaboration with other management functions in the intra-provider layer and the resource layer. Communication with the goal translation and resource management functions takes place in the intra-provider layer. An integral functionality of fault management is performance monitoring and analysis. This functionality can be provided by dedicated resource and performance monitoring functions (residing in the resource layer), but may also include implemented monitoring functionality specific to fault management tasks.

The cornerstones of the fault management concept are adaptive, scalable and effective algorithms based on probabilistic management [20]. Algorithms following the fault management concepts are to a high degree based on probabilistic approaches, providing information about the network behaviour that can be used for prediction and maintenance. Fault management algorithms cover operation within and across stacked service layers, and are typically designed for decentralized, in-network operation. The fault management algorithms make use of distributed network monitoring and performance models that gradually adjust to the variations in observed network behaviour, such that up-to-date fault management information about the network state can be efficiently provided. The fault management concept is aimed at enabling proactivity and resilience, which is crucial in both resource management and goal translation, e.g., for the seamless re-allocation of resources and maintenance of SLAs.

Resource Management Concept

Resource management functionality is implemented in the intra-provider layer. It communicates with goal translation and fault management within the layer. It communicates via the resource administration interface with resource control functions for conveying instructions for virtualisation technology and low-level resource configuration (as described in Section 2.4.1.1). It may also obtain information about the state of underlying equipment as well as monitoring data from resource and performance monitoring functions via the resource administration interface.

The resource management concept provides dynamic localization and allocation of necessary resources in an efficient and timely manner. Scalable and efficient resource scheduling mechanisms enable fast location and prioritization of available resources at a given time, ensuring short reaction times for virtual infrastructure creation and adaptation in order to minimize disruptions in service operations. Mechanisms for quick adaptation for equipment joining and leaving the pool of resources are essential for effective and dynamic allocation of resources. In large systems that may span multiple providers controlled by individual stakeholders exposing more or less limited capabilities and resource information, a homogeneous abstraction layer is required for flexible virtual infrastructure management.

Resource Control Concept

Resource control functions reside primarily in the resource layer, where it communicates with resource management in the intra-provider layer.

The concept encompasses the functionalities and protocols for virtualisation and configuration of physical resource nodes. Among the items that need to be communicated from resource level to resource management functions are capability models used in resource and capability discovery services. The resource control functions receive configuration instructions from the resource management functions.

Resource and Performance Monitoring Concept

The resource layer may implement low-level functionality for monitoring of resources and performance, which are communicated to the management functions in the intra-provider layer. The exact description of the functionalities provided is implementation dependent and may be specific for certain purposes (e.g. fault management tasks).

2.5.2 Management Functions in the Intra-Provider Layer

The state, the configuration parameters, and the capabilities of the underlying equipment may be assumed to be fully known to the intra-provider layer management functions. For resource manage-

ment this implies that the range of infrastructure service solutions that can be provided depends primarily on the current status of the equipment within the administrative domain. Similarly, fault management operations take place mainly on per-domain basis. Note that even though the intra-provider setting occurs in a single administrative domain, the physical infrastructure may span several different administrative domains, thus calling for distributed resource, fault management and performance monitoring algorithms.

In the intra-provider layer, goal translation uses (dynamically updated) performance models for the underlying equipment to select an infrastructure provisioning among a set of potential candidate solutions to provide an optimal service according to high-level service requests. The service requestor's constraints (so-called high-level goals) are translated into constraints on the performance and configurations of resources at the resource layer (so-called low-level goals), via the provided performance models. These low-level constraints are used as performance objectives by the resource management function in its communication with the resource control functions. In the intra-provider, the goal translation process can potentially find provisioning solutions that are optimized with respect to specific organizational or corporate business goals and policies.

2.5.3 Management Functions in the Inter-Provider and Service Layers

The service layer implements goal translation, and depends on information provided from the capability discovery and provider federation functions of the inter-provider layer. All these are further described in Chapter 5.

The goal translation function's primary concern is to decompose complex service requests into service requests that can be posed to suitable providers taking part in the federated service provisioning. This is done by matching the capabilities of the providers with the component requests obtained by decomposition of the original request. The available capabilities depend both on the joining peers and on the capacity declaration scheme of the federation. In turn, the capability of each provider depends on the infrastructure services it can provide, which is determined by its intra-provider layer management functions and the underlying equipment as described in the previous subsection. The inter-provider federation function provides protocols for attaching and detaching peers to the federation.

2.6 Security Aspects

The security architecture is designed to manage the current and future security challenges which may affect the CloNe infrastructure. An initial list of security challenges was based upon a well-defined and comprehensive security analysis covered in [21]. Moreover, design of the CloNe security architecture and prototyping of security modules has further modified and concretized the list of security challenges. Furthermore, the security challenges are not necessarily restricted by the use case scenarios defined for the CloNe infrastructure [4].

These security challenges in turn lead to the formulation of security goals and requirements. Security goals, with respect to the current deliverable, are considered as concrete security properties derived from the CloNe architecture. On the other hand, security requirements are considered as high-level security specifications that would be achieved by implementing a chosen set of security goals. For example, isolation, a security requirement, shall require confidentiality, privacy and integrity (security goals) for its implementation. The list of security requirements are described in Section 2.6 and the list of security goals relevant for the CloNe infrastructure are described in Section 2.6.

Security requirements

The security requirements have been obtained from the component-wise security analysis and list of security best practices from industry accepted sources, namely, CMMI, ENISA, and ITIL [22, 23, 24]. The requirements have been modified with respect to the scope of the CloNe infrastructure. Furthermore, the security requirements have evolved during the design of the CloNe security architecture and the prototyping of the CloNe security modules. This section, describes the security requirements relevant for the CloNe infrastructure and the corresponding security modules required to attain the security requirements.

Information security: The offered services and user data reside in the CloNe infrastructure and need to be protected from information leakage and misuse. Information security is achieved by successful formulation and implementation of security policies. These security policies are set and modified by the different CloNe entities.

Virtualisation management: CloNe requires an access control policy model, which enables access delegation and control of virtual resources. An authorisation logic has been developed which is capable of encoding individual delegations as access grants and using them to prove authorisation.

Isolation: In CloNe, the underlying physical infrastructure is shared by different tenants. Besides the separation of communication and the separation done by a hypervisor, the CloNe management has to take care of complying with Service Level Agreement (SLA)s of all customers.

Misuse protection: Mechanisms for detecting misuse of the CloNe infrastructure need to be devised and integrated into the overall security framework. The assurance and auditing module described in Section 4.5.3 detects and protects against misuse of the CloNe infrastructure.

Identity management: The CloNe infrastructure requires a backbone identity management framework to develop and manage identities and access control policies relevant for the different tenants and cloud users. Moreover, the different entities involved in the architecture must be authenticated, and their access to information and services should be verified against their permissible access rights.

Security goals

The CloNe security architecture requires similar security goals as any other Software as a Service (SaaS) or IaaS provisioning infrastructure, namely, availability, integrity, confidentiality, authenticity, non-repudiation, and privacy.

Availability implies that any entity, whether external or internal, is not able to affect the ability of a system to deliver services to its users in an unauthorised way. Availability is generally a quantitative metric against measurable parameters (for e.g., number of users serviced in parallel, network bandwidth, response time). In the context of security we focus on availability breaches based on attacks. In the case of CloNe this means that no user without administrative privileges on the CloNe infrastructure is able to impact the service being offered to the other users.

Integrity implies that any entity is not able to alter the data without prior authorisation; one instantiation of this is that all modifications can be detected successfully after the alteration, for example, using digital signature schemes. The integrity of communication with and inside the infrastructure elements has to be realised, so that no man-in-the-middle attacker is able to alter data that is sent to, from, or inside the cloud networking infrastructure.

Confidentiality implies that no one is able to read the data without prior authorisation. In order to read the data, the users must possess the essential credentials, such as encryption keys, as lack of credentials imply no access to the protected data. Similar to integrity, this entails confidentiality of both, the data stored inside the CloNe infrastructure, as well as the data that is transmitted during the communication between the CloNe entities.

Authenticity implies that a challenged entity can prove that their actual identity matches their

published or advertised identity. Authenticity is a strict requirement for the CloNe infrastructure in order to prove the identities of the communicating CloNe entities.

Non-Repudiation implies that any entity which has executed an action is not able to disclaim the action. For CloNe this is strongly related to traceability, i.e., to verify the geographic location of the virtual infrastructure and to verify whether it conforms to the agreed policies. Non-Repudiation is also important for accounting, auditing, and assurance purposes.

Privacy implies that an entity has control over which aspects of its personal information it wants to reveal, and to which entities. There are two primary mechanisms to enforce privacy, namely, anonymity and pseudonyms. Anonymity enables an entity to hide its identity by using a set of other identities (anonymity set), while pseudonyms enable the use of false names instead of real names. Privacy is always a trade-off between the requirements of the service provider and the cloud user. The service provider requires complete disclosure of the information that is required to provide a service. On the other hand, the cloud user wants to share as little personal information as possible.

2.6.1 Security Architecture

This section describes the CloNe security architecture which integrates the security requirements and goals described above. Previous versions of the security architecture were described in [2] and [25]. The security architecture provides quantifiable security levels within the CloNe infrastructure. This ensures that all the CloNe entities obtain a better view of real-time security levels of the service provisioning infrastructure.

The security goal translation function forms the backbone of the CloNe security architecture. The security goal translation extends the goal translation function described in Section 4.2.2 by adding security-specific functionalities. It accepts service requests, monitoring results, and topology information, and generates constraints on parameters, which are characterized with respect to the underlying resources. It also accepts a security control goal (also termed as a security objective) from the owner/issuer of the goal at a higher level in the service hierarchy, and translates it into sub-goals (which are further propagated to the lower levels in the service hierarchy), or parameters that need to be further constrained with respect to specific resources.

The security goal translation function also accepts inputs from the resource management function (for status and capabilities of the resources) and the fault management or resource monitoring functions (for the performance measurement of the resources). The security goal translation function interacts with additional security modules, in order to decompose the security relevant user requests into pareto-optimal resource specifications. These security modules include an identity management module, access control policy module, assurance and auditing module, and a Security Information and Event Management (SIEM) based intrusion detection module which uses genetic algorithm based feature selection algorithm.

These security modules can be categorized into two groups, namely, modules interacting in an intra-provider or inter-provider setting. Sections 2.6.2 and 2.6.3 provide an overview of the working of the security modules along with cross-references to places in this document where they are detailed.

2.6.2 Security Functions in the Intra-Provider Layer

This section provides an overview of the interaction of security modules in a intra-provider setting. The modules described below include identity management module, access control policy module, assurance and auditing module, and a SIEM based intrusion detection module.

Identity management module: The identity management module is responsible for managing identities and controlling access to the CloNe infrastructure. The identity management module

enables identity provisioning and authentication of the *infrastructure service users*, *infrastructure service providers*, and the virtual resources. Whenever a CloNe entity (*infrastructure service user*) wants to request a virtual resource from another entity (*infrastructure service user* or *infrastructure service provider*), it invokes the identity management module. The identity management module allows the requesting entity to automatically provision and de-provision user, provider, or resource accounts and maintains a central registry for storing the respective credentials. The identity management module uses Service Provisioning Markup Language (SPML) [26] for automatic identity provisioning and interoperation of resource provisioning requests.

Once user, provider, or resource identities have been provisioned, the identity management module allows communicating entities to authenticate each other. Authentication involves the validation or confirmation that credentials supplied by a CloNe entity are valid. The identity management module utilizes Security Assertion Markup Language (SAML) [27] in combination with standard web encryption such as Secure Socket Layer (SSL) in order to set and validate credentials. The details of the identity management module and its interaction scenarios are described in Section 4.5.4.

Access control policy module The access control module determines whether an *infrastructure service user* is permitted to access a specific resource. The resource owner (*infrastructure service user* or *infrastructure service provider*) defines an access policy that specifies which users can access the resource. Based on the access policy, user access on a resource is validated. An authorisation logic has been developed which is capable of encoding individual delegations as grants and using them to prove authorisation. The access control policy module works in tandem with the identity management module and enables fine-grained access control of the virtual resources. The details of the access control policy module and its interaction scenarios are described in Section 4.5.5.

Assurance and auditing module The assurance and auditing module is composed of two functions, namely, an assurance function which assures whether the deployed infrastructure matches the security requirements specified by the CloNe entities, and a Trusted Platform Module (TPM) based auditing function [28] which attests the geographic location of the physical resources provisioned to the *infrastructure service user*. The assurance function is built on top of the CloudAudit [29] assurance interface, while the auditing function uses XEN as hypervisor and develops a new security module for attesting the geographic location. The details of the assurance and auditing module are described in Section 4.5.3.

SIEM based intrusion detection module The SIEM based intrusion detection module is used to detect and provide real-time analysis of security alerts generated by the provisioning infrastructure. The intrusion detection system used in the module utilizes a new genetic algorithm based pre-processing algorithm and a Support Vector Machine (SVM) for detecting and classifying malware with a low false alarm rate. The SIEM based intrusion detection module supports the security goal translation function by providing it a real-time evaluation of the security level of the CloNe infrastructure. The details of the module are described in Section 4.5.1.

2.6.3 Security Functions in the Inter-Provider Layer

This section provides an overview of the working of the access control policy module in an inter-provider setting. The access control policy module enables an *infrastructure service provider* to delegate the implementation of its resources to other *infrastructure service providers* hosted outside its administrative domain. Furthermore, an *infrastructure service user* could himself be acting as a provider to other *infrastructure service users*, or *infrastructure service providers*. The detailed delegation scenarios across multiple administrative domains will be covered in Section 5.1.4.1.

3 Elaboration of the Flash Network Slice Concept

The capability to materialize the CloNe architecture in a wide range of present and future network technologies has been identified as a basic CloNe requirement [2]. This chapter addresses the network aspects of the CloNe architecture by analysing how the main components and building blocks can materialize the concept of FNS and characterizing the main service types. It also analyses how those basic building blocks can be mapped, in practice, into the most widely deployed WAN service models by service providers.

Section 3.1 identifies a set of basic CloNe architectural constraints and explains the importance of separating technology dependent and independent functions to enable the deployment of CloNe over different present and future technologies. Section 3.2 identifies the basic set of common building blocks and respective interfaces that are applicable across multiple network technologies. Section 3.3 describes how the basic FNS components can be mapped into VXDL, OCNI and Link Negotiation Protocol (LNP) protocols used by CloNe. Section 3.4 analyses how those building blocks map into concrete examples of network technologies, particularly a representative set of widely deployed network virtualisation models, i.e. IP/MPLS-based layer 2 (L2) and layer 3 (L3) VPNs. Finally, Section 3.5 shows an example including multiple administrative domains and multiple FNS components, both L2 and L3-based.

3.1 Architectural Constraints

Two basic CloNe architectural constraints are future-proofness and compatibility with the wide range of currently deployed network technologies [2]. These constraints are essential to enable technology migration and facilitate interoperability. Both require CloNe to be agnostic in relation to specific characteristics of the underlying network infrastructure. This requirement implies a clear separation between technology-independent and technology-dependent functions.

Traditionally, in a Cloud Networking environment, network domains fall in two basic groups: data centres (DC) and WAN. Usually, these two types of network domains are managed and administered by separate entities and quite different approaches are used in relation to aspects such as network protocols, resource management mechanisms, fault management and service provisioning tools. Furthermore, for both data centre and WAN specific domains, a wide range of network technologies and variants is available. Given the above-mentioned heterogeneity of network technologies, an obvious challenge is how to build a generic CloNe architecture that can be applicable regardless of the specific characteristics of the underlying network infrastructure.

A clear separation between technology-independent and technology-dependent functions, as shown in Figure 3.1, facilitates the achievement of this goal. The idea is basically that different technologies can be incorporated in the architecture simply by creating modules (represented in the figure by the blue and red arrows) adapted to the specific characteristics of each technology. This separation enables important advantages: on the one hand, CloNe is able to make use of a high number of existing network technologies, without the need to create specific "ad-hoc" solutions; on the other hand, any new technology can be accommodated simply by building the corresponding module; finally, by using common reference points, deployment of new technology can be backward compatible and interoperable with legacy technologies, i.e., a massive upgrade or replacement of equipment

is not required (no flag day). The combination of these factors should have a significantly positive impact to facilitate migration, early adoption and incremental deployment.

The WAN segment deserves a special attention in this chapter, as it provides the "glue" that binds all network components. In particular, well established network service models, namely L2 and L3 VPNs, are analysed (see Section 3.4).

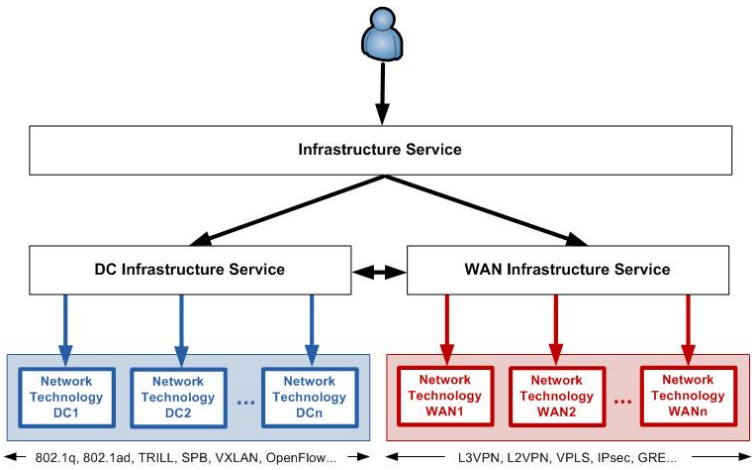


Figure 3.1: Separation of technology-independent and technology-dependent functions

3.2 Basic Components of the FNS

Once a request is made by the infrastructure service user by means of VXDL, the infrastructure service provider may decompose it and forward its multiple parts (possibly including one or multiple WAN domains) to different providers. Typically, the WAN part will be materialized as an instance of a network service, or FNS, which may take multiple forms.

The scenario represented in Figure 3.2 includes three basic network domains: customer private network, WAN and data centre. For the purposes of analysing the main networking building blocks in Cloud-based services, the characteristics of the customer private network play a relatively minor role and can be seen as part of the WAN managed service (e.g. L2 or L3 VPN).

Figure 3.2 illustrates the case where infrastructure of two customers (1 and 2, identified by two different colours) is distributed across private network and Data Centres DC1, DC2 and DC3. The basic components, which are identified below, are illustrated in the figure. The WAN service, in several possible forms, constitutes the glue that binds together DCs and customer premises. Therefore, the establishment of an FNS strongly relies on the underlying technology deployed in the WAN.

- WAN FNS: provides interconnection of remote customer premises, as well as between customer premises and data centres, or between data centres. A WAN FNS can be accessed through a number of points of presence, or ports in CloNe terminology, which are typically materialized by an edge device. The set of ports define the geographical footprint of the FNS (e.g. regional, national, global). A WAN service must be capable of supporting multiple customer service instances and guarantee isolation among them. Depending on the service type, this can be materialized in one of multiple forms, e.g. virtual router, virtual switch, VRF (Virtual Routing Forwarding), VSI (Virtual Switching Instance). From the customer

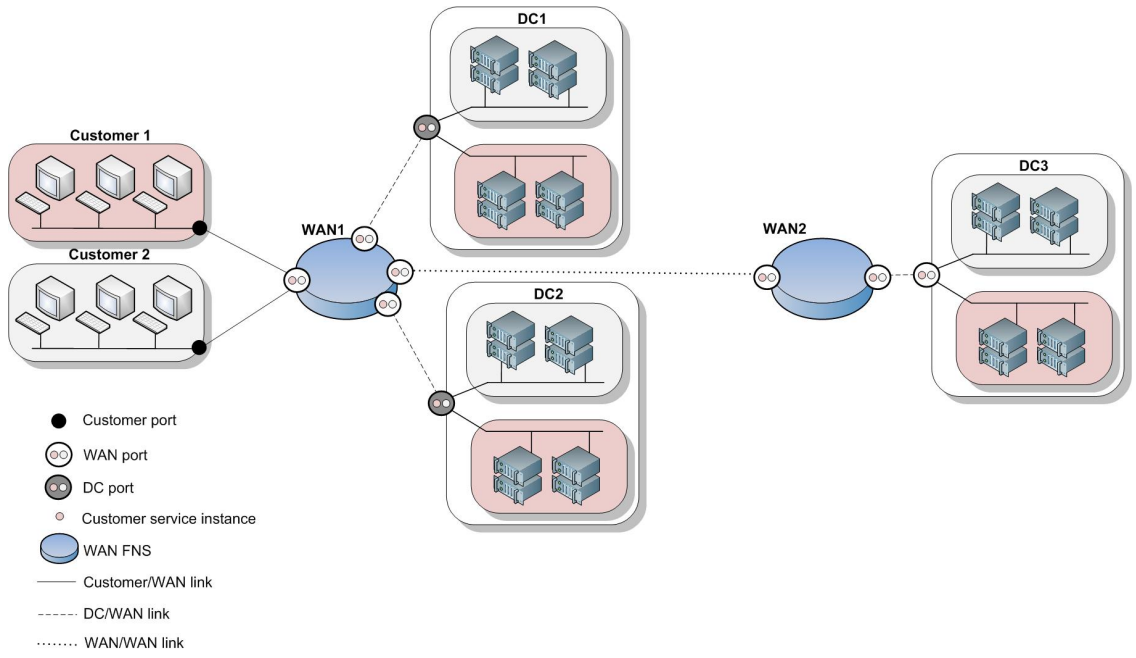


Figure 3.2: Network model building blocks

point of view this service instance is typically seen as a dedicated L2 or L3 device (depending on the type of service offered by the service provider).

- Data Centre Network: Several design approaches can be used in the data centre. In the case of Cloud Service Providers, multi-tenancy is a basic requirement, which implies that some form of network virtualisation (e.g., 802.1q (VLAN), Q-in-Q, VXLAN) is needed. A data centre is connected to the outside world by a number of ports, which basically correspond to a demarcation point between Data Centre and WAN infrastructure. At the port level, adaptation is usually performed between DC and WAN domains; a DC may be connected to one or more WAN ports (e.g. for redundancy reasons). A data centre port must support multiple isolated service instances (one per tenant) to allow multi-tenancy, which presupposes that some form of network virtualisation (e.g., VLAN) is used inside the DC.

- DC/WAN link: corresponds to the connection between the Data Centre and the WAN FNS. In order to enable DC multi-tenancy and guarantee isolation between different customers, multiple virtual links are typically multiplexed over this physical link. There can be more than one connection between a DC and a WAN service provider and a single DC may be connected to multiple WAN service providers.

  Figure 3.3 illustrates this from the perspective of one data centre. In this example there are three links, two going to *WAN 1* and the third going to *WAN 2*. In the figure, these links are referred to as *service provider logical links* (SPLL). The SPLLs are split into a number of *tenant logical links* (TLL), each illustrated with a unique colour.

  By agreeing in advance on a unique SPLL identifier for each SPLL, the DC and WAN service providers can negotiate which SPLL to use when a FNS shall be instantiated on-demand for a certain tenant. As part of such a negotiation the TLL will be created and given a TLL identifier for later reference (e.g., if its properties should be changed). Hence there is a one-to-one mapping between a certain FNS and a TLL. The LNP was designed in order to automatically perform this negotiation.

- WAN/WAN link: link between two WAN providers. Typically each WAN/WAN link carries multiple customer virtual links. Identification of virtual links must be negotiated between peer network providers. LNP can be used for negotiation in this context as well. A detailed discussion of inter-provider WAN/WAN issues is for further study.

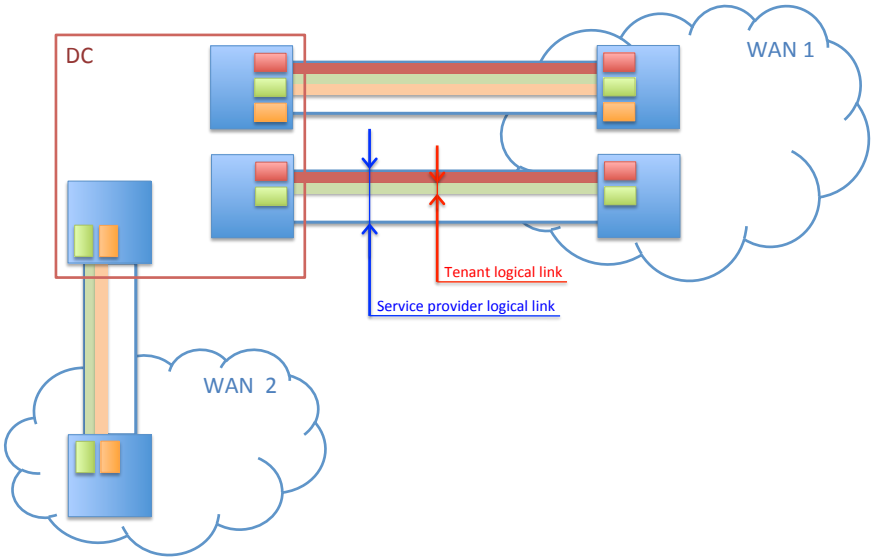


Figure 3.3: The DC/WAN interface for tenant isolation.

3.3 Mapping FNS components to CloNe protocols

The purpose of this section is to show how the basic FNS components are mapped into the CloNe protocols. In this section only a general overview is provided. Further details, as well as information on prototype implementation, can be found in [3].

3.3.1 VXDL and OCNI

VXDL and OCNI represent two emerging standards in the cloud networking space that we are actively evolving as part of CloNe. Both are capable of modelling the FNS concept.

VXDL has high-level representation of both the virtual infrastructure resources and the constraints attached to them. Originally targeted at network description, it now includes complete infrastructure description. It can be used to describe goals, objectives and time lines, suitable as input for driving the inference of concrete infrastructure configurations.

OCNI is an extension to OCCI, an existing cloud interface standard. In contrast to VXDL, its origins lie in cloud computing and the OCNI extension represents an improvement in its treatment of virtual networks. It can be implemented directly on several existing infrastructure managers, including OpenNebula and OpenStack, making it suitable for describing the concrete infrastructure configurations.

These standards overlap in application but also demonstrate different strengths. Table 3.1 shows a mapping between our infrastructure concepts and classes in the VXDL and OCNI data models.

We use both in our prototyping: VXDL for high level, goal based, descriptions that are interpreted and translated to OCNI for concrete input to infrastructure managers.

Table 3.1: Mapping CloNe concepts to VXDL and OCNI

Concept	OCNI	VXDL
Resource	Resource	vResource
Compute	Compute	vNode
Storage	Storage	vStorage
FNS	FNS	vRouter
StorageInterface	StorageLink	Defined by a vlink between the vStorage and a vNode
NetworkInterface	NetworkInterface	Defined by the vLink
Infrastructure	Resource + Infrastructure Mixin	virtualInfrastructure
Infrastructure Service Provider	Resource + Infrastructure Service provider Mixin	A vAccessPoint connecting two virtualInfrastructures
Port	Network + Port mixin	Defined through the vLink
Link	NetNetLink	vLink
RemoteAttachment	Port + RemoteAttachement Mixin	Could be any vResource
NetworkConstraint	NetworkConstraint Mixin	The specification of a network is done as vLinks, vRouters
NetworkFunction	Port + NetworkFunction Mixin	The specialization of a vAccessPoint, a vRouter

3.3.2 Inter-Provider Coordination in CloNe: DCP

The DCPs LNP, defined in the prototyping deliverable [3], is responsible for the coupling of different network managed services. In other words, and looking at Figure 3.2, the protocol is responsible for the establishment of, for example, the virtual links between WAN1 and WAN2 and WAN1 and DC1 for both costumers. This coupling may be materialized through different network technologies (L2 or L3), such as VLANs, Internet Protocol Security (IPsec), Generic Routing Encapsulation (GRE) or MACinMAC, requiring the exchange of information between the involved providers to agree on specific configurations.

The LNP is a technology-independent protocol in the sense that it provides the means to enable the technology-dependent information exchange independently of the technology. The technology-dependent information is embedded in the *encap_scheme* parameter of the LNP information structure. The information carried within the *encap_scheme* is divided in two parts, the *encap_type* and the *attributes*. The former identifies the network technology (i.e. encapsulation scheme) (e.g. VLAN) and the latter carries the specific configurations/attributes (e.g. VLAN id) that need to be exchange depending on the technology. Table 3.2 presents a mapping of possible network technologies with the information carried within the LNP. For a deeper understanding of the LNP the reader is advised to consult [3].

3.4 Materializing FNS in the WAN

The ability to match the dynamism and elasticity of the Cloud is a basic requirement of the network infrastructure to build Cloud services and applications. The concept of FNS is expected to address this requirement by enabling the automatic establishment of network resources in a time

Table 3.2: Link Negotiation Protocol: technology dependent information

[gray]0.9**Technology**	[gray]0.9***encap_type***	[gray]0.9***attributes***
VLAN	VLAN	*vlan_id*
GRE	GRE	*tunnel_source*, *tunnel_destination*, *tunnel_key*
IPsec	IPSEC	*tunnel_source*, *tunnel_destination*, *encryption_type*, *hash_type*, *authentication_method*, *group*, *transform-set*, emphlifetime

frame comparable with existing virtual infrastructure computing and storage resources. OpenFlow and SDN-based technologies have recently emerged as promising solutions to handle this kind of requirements by fully decoupling the control and management plane from the data plane. However, in the foreseeable future, access to cloud by most enterprise networks will likely be based on managed WAN services such as L2 or L3 VPNs, which at the moment lack dynamic and elastic properties of cloud services, as they were tailored for relatively static networks and were not conceived to cope with such requirements as on-demand provisioning, elasticity and resource mobility. In the long term, novel solutions enabling more dynamic control of network resources, as well as better integration with service and application requirements are likely to gain wide acceptance. However, given the huge installed base of legacy technology, a significant change is not expected to take place soon. It is clear that L2 and L3 VPNs will continue to be the basis of FNS in the WAN in the foreseeable future. This section addresses the issue of how the FNS concept can be materialized by the currently dominant models for enterprise services, namely L2 and L3 VPNs.

In the network reference model of the CloNe architecture, the DC (network) infrastructure and WAN infrastructure services expose well familiar forwarding functionality of Ethernet frames or IP packets towards the cloud tenants. Request of these services is done using a technology independent northbound RESTful interface typical of cloud infrastructure services. For instantiation of the infrastructure service, a technology dependent interface is used.

Usually the taxonomy for WAN network services is based on the OSI layer used to build it, which basically leads to a division of layer 3 and layer 2 service types. The most widespread model of layer 3 services is BGP/MPLS IP VPN (as per RFC 4364 [30]). As for layer 2 services a wider range of variants exist. The Metro Ethernet Forum has defined two basic types for Ethernet-based services: Ethernet Line (E-line, also known as Virtual Leased Line) and Ethernet LAN (E-LAN, or Virtual Private LAN Service, VPLS, in IETF terminology) [31][32][33]. A brief description of the three basic service models is provided below.

In a layer 2 FNS, a network interface may materialize as a physical interface (i.e., an Ethernet card), a virtual one (such as those associated with virtual machines) or a logical interface typically associated with IPSec, GRE and similar tunnels or aggregated physical interfaces. A port may attached to a *RemoteAttachment*, which is the local endpoint of layer 2 WAN communication service. The establishment of a RemoteAttachment can be done statically in advance or dynamically on-demand. In the latter case, the DCP plays a central role, as elaborated in Section 3.3.2.

The L3 FNS provides a routing and forwarding function that can be associated with one or several L2 FNS. It can also be associated with the local and tenant specific attachment point of a layer 3 WAN communication service (see further Section 3.2). Analogous to L2 case, such attachment points can be established statically in advance or dynamically on-demand. Again, in the latter case DCP plays a key role.

In the following, the materialization of the FNS concept, based on some of the most widespread models of L2 and L3 network services, is presented.

- IP VPN (L3VPN): each instance of the service corresponds to a private IP network. Multiple private IP networks share a common infrastructure, potentially using overlapping IP address spaces. Each provider edge router typically contains one VRF (Virtual Routing Forwarding)

per VPN, which can be seen as a VPN-specific IP routing table and corresponds to the basic service instances. If a L3VPN is used to interconnect DCs, or connect customers to private subnets inside DCs, multiple service instances of some form (e.g. VRF-lite) should be configured at the edge node of the DC to segregate traffic belonging to different customers. The customer subnet inside the data centre can be seen as an extension of the customer private network. Because the service is based on layer 3 protocols, and the DC-WAN interface is L3-based, any L2-specific information in this interface (e.g. VLAN ID) has local meaning only. The IETF specification can be found in [30].

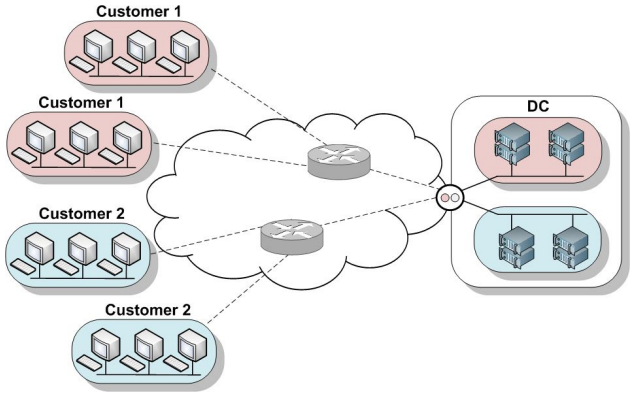


Figure 3.4: Layer 3 VPN

- Virtual Leased Line (VLL): is a layer 2 service, functionally equivalent to a point-to-point Ethernet Virtual Connection, or a TDM private line. Optionally, service multiplexing may be provided, in which case more than one VLL service instance is offered on the same customer physical interface. Customer VLAN tags may optionally be preserved. Figure 3.5 illustrates the case where two customer private networks are connected to a data centre (other scenarios making use of a VLL service, e.g. data centre interconnection, could be considered as well). Since this is a L2-based service, L2-specific information such as VLAN ID may, or may not (depending on the service characteristics), be preserved across the WAN [33].

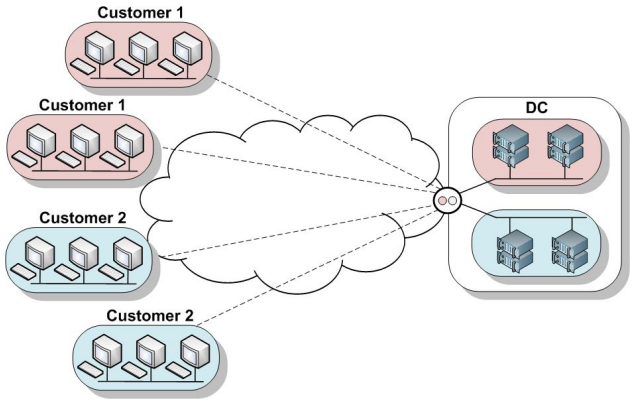


Figure 3.5: Virtual Leased Line

- Virtual Private LAN Service (VPLS): provides multipoint connectivity between two or more sites (e.g. DC, customer private site) and emulates LAN service across a WAN. The service

is supposed to mimic the behaviour of an Ethernet bridge, including dynamic MAC address learning, flooding and support of Spanning Tree Protocol, which implies a limited scalability of the service (e.g. number of participating MAC addresses). In the example illustrated in Figure 3.6 an operator provides two instances of the VPLS service. The VPLS service, in two different flavours, has been specified by IETF [34] [35]. An alternative term to designate essentially the same concept, proposed by the Metro Ethernet Forum, is E-LAN [33].

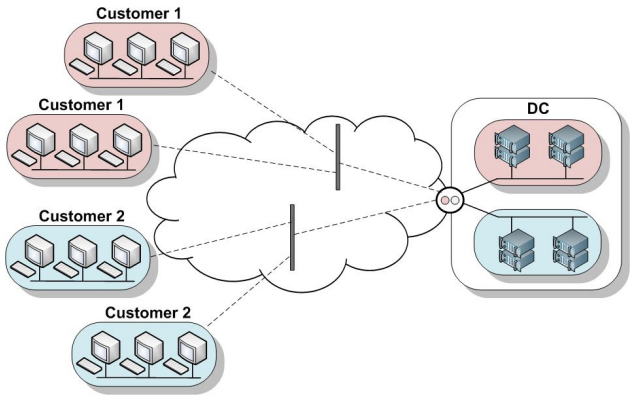


Figure 3.6: VPLS (E-LAN)

Table 3.3 summarizes how the basic components identified in Section 3.2 can be mapped into these network service models.

3.5 Virtual Infrastructure Example

In most practical Cloud Networking scenarios, end-to-end connectivity is provided by multiple FNSs, based on different virtual network domains, possibly built on different technologies and protocol layers. Figure 3.7 exemplifies a virtual infrastructure composed of three cloud data centres DC1, DC2, and, DC3, along with two wide area networks WAN1 and WAN2, all of which are capable of delivering FNS services. The Internet is also shown, in the lower right part of the figure.

On the left side of the figure, it can be seen that in DC1, the tenant's virtual infrastructure consists of two virtual L2 FNSs to which one VM and two VMs are attached, respectively. The layer 2 virtual network that corresponds to the L2 FNS with only one VM in DC1 spans across to DC2 using a FNS provided by WAN1. There, two more VMs are attached to the same virtual layer 2 network domain. Thus, a single layer 2 domain is composed of three different FNSs located in DC1, WAN1 and DC2.

In DC3 there are two more L2 FNSs, one attaching two VMs and the other only one VM. Between VMs on the same virtual L2 FNS, IP packets can be exchanged without the need of routing functionality.

The connection between different FNSs is provided by tenant-specific logical links (TLL, as defined in Section 3.2). In the example illustrated in Figure 3.7, the L3 FNS in DC1 connects to the L3 FNS in the WAN via a TLL assigned to the tenant, which in turn provides connectivity to the left-hand side L3 FNS in DC3 via a second TLL. That L3 FNS, in turn, is connected to one of the L2 networks in DC3. Assuming that appropriate routing entries have been injected in all involved L3 FNSs, this means that VMs attached to different L2 FNSs in DC1 and DC3 can communicate to each other.

DC3 also contains a L3 FNS with Network Address Translation (NAT) capability. Through that L3 FNS VMs can reach the Internet, and all IP packets originated by those VMs will enter the

Table 3.3: Components, relation to CloNe architecture and mapping to standard network services

Component	**Relation to CloNe**	**L3VPN**	**Virtual Leased Line**	**VPLS**
FNS / WAN service	Service provided by network operator, requested by OCNI at provisioning time	IP/MPLS VPN (RFC 4364)	VLL/Pseudo-Wire (RFC 3985)	VPLS (RFCs 4761, 4762)
WAN Port	WAN infrastructure ingress point, requires (re)configuration whenever a new service request by OCNI is to be enforced e.g., create a new customer service instance (e.g. new VRF, in the case of L3VPN)	Located at edge router supporting L3VPN PE functionality; associated with VRF	Located at edge router as Pseudo-Wire endpoint	Located at edge router supporting VPLS PE functionality
Data Centre Port	To be reconfigured per customer request (by means of VXDL) at service provisioning time	Tenant traffic isolation by means of virtual router, VRF lite	Some form of network virtualisation/encapsulation (e.g. 802.1q, QinQ, MACinMAC, MPLS, L2TP, GRE)	Some form of network virtualisation/encapsulation (e.g. 802.1q, QinQ, MACinMAC, MPLS, L2TP, GRE)
Customer service instance at WAN Port	Created per customer request (on WAN side) at service provisioning time; specific parameters (e.g. IP addresses) are negotiated by DCP/LNP	Virtual Routing Forwarding (VRF)	Pseudo-Wire endpoint	Virtual Switch Instance (VSI)
DC / WAN logical link	Specific characteristics (e.g. encapsulation type) negotiated by DCP/LNP	IP addresses, Routing protocol, L2 Encapsulation type (e.g. 802.1q, QinQ, MACinMAC, MPLS, L2TP, GRE)	Encapsulation type (e.g. 802.1q, QinQ, MACinMAC, GRE, MPLS), Encapsulation ID	Encapsulation type (e.g. 802.1q, QinQ, MACinMAC, GRE, MPLS), Encapsulation ID
WAN / WAN logical link	Specific characteristics negotiated by DCP/LNP	Several options available (specified in RFC4364)	Inter-AS PW	Inter-AS Tunnel
(Typical) Constraints	Requested by customer on-demand or statically defined by provider	Access bandwidth, Routing protocol (PE-CE), maximum delay, jitter, packet loss, firewall rules, packet classification rules	Bandwidth profile (e.g. CIR, EIR, CBS), service multiplexing, maximum delay, jitter, packet loss	Bandwidth profile (e.g. CIR, EIR, CBS), Maximum number of MAC addresses, service multiplexing, maximum delay, jitter, packet loss

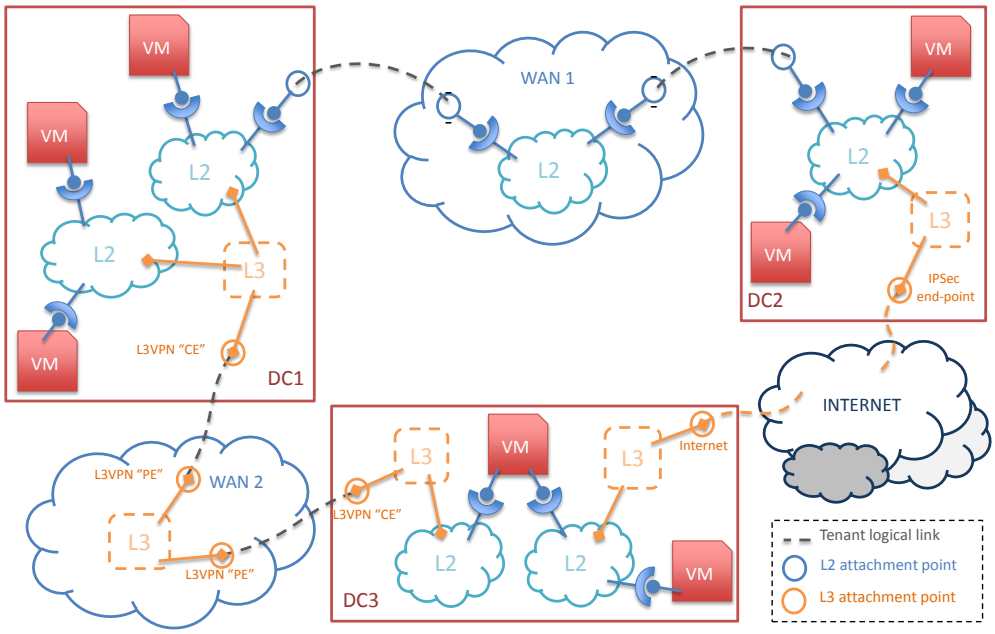


Figure 3.7: A complex virtual infrastructure for one tenant with multiple FNSs

Internet with NATed source IP address. In DC2, an L3 FNS with one of its two attachment points attached to a local IPSec tunnel end point can be seen. It means that packets routed by the L3 FNS to that destination will be injected into that IPSec tunnel and will traverse the Internet to some remote destination (not shown in the figure).

4 Elaboration of the Intra-Provider Layer

This chapter focuses on the CloNe management and security architectures with respect to services implemented within a single provider. In particular, the chapter presents a number of implemented approaches, toward addressing specific management and security challenges that are significant for meeting the requirements, and for realizing the vision of CloNe. As management of inter-provider virtual infrastructures to a great extent is done through management of its single provider virtual infrastructure components, the material presented in this chapter has bearing also on the inter-provider chapter (Chapter 5).

Recall from Chapter 2 that an intra-provider infrastructure is a number of virtual resources managed collectively within a single administrative domain, where a single administrative authority has management rights over the underlying equipment and virtualization technology. The management of an intra-provider infrastructure is divided into four main tasks. The first task (goal translation) is to map service requests to optimal resource provisioning solutions, i.e., an optimal selection of virtual infrastructure provisions. The second task (resource management) is to define the mapping between such virtual infrastructure provisions and the underlying equipment. The third task (fault management) is to ensure stability with respect to SLAs and performance by mitigation of faults and degradations through detection and localization of anomalous behaviour. The fourth task (security management) concerns ensuring that the infrastructure is provided according to security and privacy requirements. The first three of these tasks have been addressed in the management task of WPD, while the fourth task has been addressed in the security task of WPD.

The four tasks rely on supporting services, including communication, resource monitoring, performance and resource modelling, and resource controllers. In particular, the functions performing the tasks within the intra-provider layer rely on communication with the other architecture layers. For obtaining performance and resource monitoring data, as well as for controlling the underlying equipment, the management functions connect to the resource administration interface. In particular, the resource management functions may rely on status and capability information provided via the resource administration interface. For receiving service requests, the goal translation function needs to interface with the service and inter-provider layers. While monitoring functionalities may be implemented in the resource layer, it is not excluded that intra-provider management functions implement monitoring functionalities that are specific to their operations. For example, fault management may build statistical performance models through active probing, obtaining monitoring data that not necessarily is provided by the resource layer functionalities.

4.1 Management Functions—Operations and Interactions

The primary functions in the intra-provider domain implement the concepts of goal translation, fault management, and resource management. These functions are called goal translation function (GT), fault management function (FM), and resource management function (RM). Each function consists of several blocks, together providing the necessary functionalities for intra-provider management. The exact configuration of functional blocks is implementation dependent and it is not crucial at this level of description to fix these implementation considerations. The management functions can be instantiated in any virtual instance of an infrastructure service, given that the infrastructure service provider implements it according to its specific management purposes and

with respect to the available infrastructure.

This section briefly describes the functions with respect to their functional blocks and their communication and interaction behaviour, both with other management functions and with other parts of intra-provider infrastructure. For more detailed descriptions, please refer to the earlier version of the management architecture [2].

Goal translation function

GT consists of four blocks: *high-level objectives block*, *optimization block*, *monitoring block*, and *low-level objectives block*. The high-level objectives block receives and processes high-level infrastructure and security objectives from the infrastructure service interface. The optimization block consists of a multi-objective optimization algorithm, transforming high-level objectives into performance objectives, which in turn are stored in the low-level objectives block. The monitoring block receives monitoring data from the monitoring functions (as well as possibly richer performance models from FM) relevant for determining whether the performance objectives are fulfilled. Observe that the monitoring block not necessarily implements full resource and performance monitoring functionality. The monitoring block may rather merely store information about which monitoring data is relevant for determining whether a certain low-level objective is breached, and rely on monitoring data from dedicated management functions.

Apart from service requests, GT needs as input, for each service request, a set of candidate provisioning solutions, and relevant, up-to-date, performance models to use for the optimization procedure. The provisioning solutions selected through the optimization process are communicated, as output, as performance objectives for the resource manager.

GT operates in two processes: the optimization process and the monitoring process. The optimization process continuously tries to find optimal resource provisioning solutions to goals that are yet to be fulfilled (which may be new service requests or violated goals that have been triggered for re-optimization). The monitoring process keeps track of the low-level goals and either triggers re-optimization of high-level goals or declares a high-level goal fulfilled, depending on whether corresponding low-level goals are fulfilled.

Fault management function

The responsibility of the FM function is to monitor the behaviour or state of resources, and report disturbances to goal translation, resource management, security mechanisms, infrastructure service users, or infrastructure service providers in collaboration. The FM function consists of two functional blocks — "Performance and root cause analysis" (PRA) block and the "Detection and localization" (DL) block. Algorithms that map to the functional blocks operate using observed network measurements and logged events as input, and may receive and process data from the monitoring tools implemented in the Resource and Performance Monitoring block (RPM), for evaluation of the resource behaviour (based on measurement requests). Measurement requests to specific resources can be performed through the management interfaces. The FM function provides 1) disturbance information, such as detected and localized faults, anomalies, and changes in the observed behaviour of monitored resources; 2) potential root-causes; and 3) probabilistic models and analysis of the long-term behaviour of the monitored resources, in terms of e.g. failures and correlated events. FM information about disturbances and changes can be communicated to RM and GT for fault handling, re-configuration purposes or when a service is initiated and deployed. The models created for FM analysis might also be useful for RM and GT operations in addition to RPM data (e.g. for resource allocation). FM algorithms can implement functionalities mapped to either in PRA, DL or both, and can be based on information exchanges between these blocks.

An FM-algorithm might use already existing performance monitoring tools and models from the RPM, or might implement algorithms in both FM and RPM for specific FM purposes.

The overall operation of the FM function can be described in terms of four main processes: FM monitoring/modelling and detection, fault isolation, root cause analysis, and fault handling processes. The first process is triggered by the RM process, whenever resources is allocated and configured. The FM algorithms are configured according to specification by the RM. The FM algorithms continuously process monitoring data and measurements over the equipment in each virtual infrastructure in a decentralized manner, running on the virtual resources. The models obtained in this process are then used for detection of faults and performance degradations, for adaptation of FM algorithmic behaviour, or as richer input to other management functions [20]. The fault isolation process is triggered when failure or performance degradation is detected for a virtual resource within a virtual infrastructure based on processed measurements. The fault is localized in a distributed and collaborative manner within the virtual infrastructure. The root cause analysis process is triggered when a fault has been localized. As the cause can originate from some other parts of the cloud, the root cause analysis process correlates events from virtual infrastructures running on shared resources for the purpose of analysing the order of events and isolating the true origin of the fault. The fault handling process is triggered when the root cause has been determined. In collaboration with the RM function, actions to resolve the problem are taken.

Resource management function

RM manages provisioning, configuration and availability of storage, network, and computation resources within its management scope. It maps virtual infrastructure requests to the physical resources of the cloud infrastructure in order to make sure that the provider's goal with respect to how its physical resources are allocated is satisfied while ensuring that low-level goals associated with the requests are achieved. Given a request for virtual infrastructure, RM identifies possible solutions based on input from Resource and Performance monitoring, status reports from the FM function, as well as goal fulfilment requests from the GT function. RM implements a solution selected by GT, which is dynamically adapted in order to handle varying conditions in the cloud. RM function allocates/de-allocates multiple resources of multiple types at the same time, allowing deployment of entire, or parts of a virtual infrastructure request.

The RM function consists of two blocks; resource allocation (RA) and resource adaptation-optimization (RAO). The RA block is involved in the initial deployment of a virtual infrastructure request: computing possible solutions for a request and implementing the one chosen by GT. The RAO is a reactive process that dynamically re-optimizes an existing resource allocation in order to make sure that the goals of the provider as well as the low-level goals of the customer are achieved at all times.

The RM function needs as input: information on the physical resources (such as type/architecture, resource capacity, topology, current load, performance objectives, etc.), virtual resources (such as demand forecasts, low-level goals, etc) and faults (performance degradation and disturbance information). As output, RM provides resource availability information on request; possible solutions for provisioning (to the GT optimization block); monitoring specification for resources and configurations; parameter settings to the resource control functions for configuration of resources and equipment included in the accepted provisioning solution.

4.2 Implemented Approaches to Goal Translation

In this and the following three sections follow descriptions of the approaches made within the management and security tasks of WPD toward realizing the CloNe vision and requirements with respect to the management and security architectures in the intra-provider setting. First out are the approaches to goal translation within a single provider.

For the intra-provider, we present three approaches to address the challenges for goal translation. The first challenge is to enable the specification and management of service requests independently of the underlying physical or virtual infrastructure. Among the benefits that may result from addressing this challenge is that service requesters may use a high-level specification language both for specifying and for defining constraints on the requested service, from which a virtual infrastructure service can be provided through automatic mapping of the high-level specification to virtual resources. Another benefit lies in the possibility of enabling desirable features of the mapping mechanism through extension of the high-level request language. The challenge of devising such a language and mappings is relevant for both the intra- and inter-provider settings. While the language for requesting services should not necessarily be different, the mapping from service request to virtual services may take different forms in the intra- and inter-provider cases. The approach to addressing this challenge for the intra-provider case is described in Section 4.2.1.

The second approach addresses the challenge with respect to matching of service requests a provided infrastructure services taking into account the volatility and inherent uncertainty in a distributed virtual infrastructure environment. With the multi-tenant and multi-provider requirement of CloNe, together with the distributed nature of cloud networking, it may become hard to predict and determine an infrastructure provision that is appropriate for a particular service request for the duration of the service. This has implications on how services should be specified and on how to best provision for the service request, as well as for conforming to SLAs related to the request. This challenge has been addressed via a proposed framework where services are requested in terms of the desired performance (instead of e.g. desired configurations of VMs) and where such specification are translated, through optimization, into performance objectives, where the translation takes the current state of the physical and virtual infrastructure into account when determining an optimal provision. See Section 4.2.2.

The third approach concerns how high-level security requirements (specified by users or tenants) can by translated into resource-level specifications of security policies. This approach is described in the security management section (Section 4.5) further below.

4.2.1 VXDL as a Language for High-level Goals

To address the challenge of devising a suitable high-level request language, we propose the use of VXDL [36]. VXDL is a unifying modelling language for describing virtual infrastructures, which defines a simple grammar for enabling a high-level representation of both the diversity of virtual resources (e.g., nodes, routers, access-points, links, storage) and the constraints attached to them (e.g., capacity and performance attributes, reliability and security levels, geo-location of resources, network topology). With VXDL, end users can detail a FNS configuration independently of the physical substrate. A benefit of VXDL is that it can be extended with mechanisms for controlling the mapping of user requests to resource allocation. This subsection presents a VXDL [36] extension on Elasticity that provides a way to define rules to react in a typical way (*e.g.* changing the resources' capacities/attributes, adding new resources dynamically) when a given event is detected (*e.g.* triggers over monitoring events on classic or custom metrics).

The state of the art with respect to defining elasticity rules and providing for automated scalability [37, 38, 39, 40, 41, 42] considers only scaling arrays of virtual machines up and down, according to some metrics. These metrics are usually related to CPU, memory, storage and network activities.

Scaling arrays of VMs up and down are only part of the elastic provisioning. Moreover, considering VMs as the granularity unit for scaling virtual infrastructures can lead to a waste of capacity for both user and provider. By considering elasticity on network resources as well as computing resources, the approach extends the state of the art to a more fine-grained mapping of user requests to resource allocation.

For the purpose of meeting the challenges of CloNe, it was identified that VXDL must be ensured to be able to express constraints on the following elements:

Resource capacities: (*e.g.* CPU, memory, storage, bandwidth configuration) we introduce the intervals [MIN, MAX] to indicate the values of capacity that are acceptable for a given resource (only values belonging to interval should be provisioned). We also define events to describe the conditions which would cause a value to scale.

Scaling virtual elements: the virtual array element also get a interval [MIN, MAX] to specify the limits in terms of elements inside the array.

Latency: the latency attribute also gets a [MIN, MAX] interval but its meaning is slightly different from the other attributes. It provides additional information that can be used to identify the correct position/place where resources must be provisioned so as to respect this constraint.

Location: this allows to identify a specific (or general) location where a resource (or a complete FNS) should be provisioned.

Application performance and metrics: by using the TAGS system (a triplet composed of a key, a type and a value), a user can identify metrics that should be monitored during the execution. A set (or subset) of metrics can be combined for representing the appliance performance.

The extension of VXDL, with the constraint types listed above, allows to define elastic rules that join events in the virtual infrastructure with actions to be executed. These events can be defined as thresholds over monitored resources. The actions permit to modify any element, or attributes of any element, that are defined within a virtual infrastructure. The actual execution of the modification may depend on the availability of the physical resources or the capacity of the physical resources to allow runtime modifications of the initial deployment.

The Elastic Control API is designed to allow infrastructure service users or tenants to interact with an elastic FNS service. To be able to generate the events or to identify that a given event is triggered, we assume that we have access to a mapping of the monitoring metrics over the elements of the virtual infrastructure (possibly over a REST API) of the CPU, Memory, storage and bandwidth metrics.

Table 4.1 summarizes the different actions possible through the Elastic API. These actions can be exposed through a REST API. For instance, the creation could be done by doing a POST operation on the specific element of the virtual infrastructure, as in *POST /ypxis/{ypxiId}/ylinks.vxdl*, which would allow to create a new virtual link. The described extension of VXDL has been validated through inclusion in the CloNe prototype.

4.2.2 Goal Translation Algorithm

The volatility and inherent uncertainty in cloud networks has potentially negative implications for service provisioning and SLA conformance. To tackle this, an algorithm, called *goal translation algorithm* (GTA), was formulated where the mapping of service requests to virtual infrastructures is made via optimization taking the state of the underlying resources into account.

In the intra-provider, GTA transforms constraints (called goals) on service performance parameters (which in the general case are QoS parameters, and in the degenerate case merely VM

Operations	Descriptions	Parameters
Get	Get all operations allowed for a resource.	UUID of a resource
	Get [MIN, MAX] physical information.	
Create	Create a VResource (VNode, VRouter, VLink, VStorage, VAccessPoint)	VXDL file
Delete	Delete a VResource	UUID of a resource
Update	Update a VResource configuration	Specific configuration

Table 4.1: Elastic API possible actions

specifications) into resource configuration objectives. In this process, GTA first collects potential solutions (i.e. resource configuration specifications) from one or several resource managers. The potential solutions take into account the current state of the providing nodes, obtained via performance and resource monitoring function. A potential solution gives an estimate of the level to which it can perform with respect to the performance parameters given by the goals. With the aid of performance models, GTA selects the candidate solution that best matches the requested goals. The best match is determined through multi-objective optimization. The resource manager that provided the best candidate solution is subsequently offered to provide the service, according to configuration objectives obtained from the performance model with which it supplied the solution alternative. Next follows a brief description of a basic instantiation of the GTA framework, for which details can be found in [43].

In [43], we consider the case with one goal for one requested service. The goal consists of the constraints $C_1, \ldots, C_n$ where each constraint C_i is a probability distributions over a performance parameter ξ_i. It is assumed that there are m possible provider nodes for the service and that exactly one provider will suffice. When posed with the goal, GTA requests potential solutions from each node, thus obtaining the performance guarantees $Q_{j,i}$, for $j = 1, \ldots, m$ and $i = 1, \ldots, n$. $Q_{j,i}$ is a probability distribution describing how well the node j can perform with respect to the parameter ξ_i. Via a definition of distance M between probability distributions, we can for the intra-provider choose the $Q_{j,i}$ that is the closest to C_i for each i, via minimizing the expression $\sum_{i=1}^{n} |M_i(Q_{j,i}, C_i)|$ with respect to $j = 1, \ldots, m$.

$$\min_j \sum_{i=1}^{n} |M_i(Q_{j,i}, C_i)|, \qquad j = 1, \ldots, m \tag{4.1}$$

In the intra-provider, we can assume that each node provides an estimate $Q_{j,i}$ as close as possible to C_i, since that (as argued in [43]) leads to the most efficient resource allocations. However, in the inter-provider setting, with possible competition and diversity of business models, we need a slightly different optimization procedure, as we cannot assume that $Q_{j,i}$ reflects the true capacity of node j as there may be economic, or other, incentives to underrepresent (or over represent for that matter) ones capacity. See [43] for further details on such issues in the inter-provider. Central in this approach are the performance models from which the performance guarantees $Q_{j,i}$ are obtained. Each service provider $j \in \{1, ..., m\}$ is assumed to have been monitoring its behaviour with respect to $\xi_1, \ldots, \xi_n$ and its own service configuration parameters $\pi_{j,1}, \ldots, \pi_{j,m}$, and thus obtained probabilistic performance models of the form $P(\pi_{j,1}, \ldots, \pi_{j,m} | \xi_{j,i})$ Note that we may assume that $\pi_{j,1}, \ldots, \pi_{j,m}$ are parameters of distributions that describe the configuration of the node. Observe that via such models, parameters used in the higher level goals (i.e., the parameters

ξ_i) are related to parameters $\pi_{j,1}, \ldots, \pi_{j,m}$ used in performance objectives (or lower level goals). Assuming that it is k that minimizes Equation 4.1, i.e., that the performance guarantees provided by node k were the closest to the constraints C_i, then the node k will be selected for provisioning according to the request. And thus according to the performance model, node k will operate under the objectives given by $\pi_{k,1}, \ldots, \pi_{k,m}$.

4.3 Implemented Approaches to Fault Management

Fault management in complex, virtualized and dynamic networks requires solutions that can adapt to varying virtual topologies and effectively exchange information between fault monitoring entities. In addition, fault management algorithms operating in two dimensions of the cloud networking infrastructure are also needed. First, individual services that span across the network need to be monitored and analysed for detection of performance degradations and faults. Secondly, fault management algorithms that operate across stacked layers are needed for efficient fault localization and root-cause analysis within relevant sub-topologies. The challenges in these respects therefore relate not only to scalable protocols, but also to timing and synchronization in the process of e.g. correlating detected events. Another challenge is the limitations and restrictions in the actual hardware and software across heterogeneous equipment, controlling to which degree a network entity can be monitored for fault management purposes. In connections that share passive, forwarding network equipment, direct detection of performance degradations and localization to intermediate connections is rather complicated, as measurements of e.g. link delays only reflect the behaviour of the entire connection between the active, endpoint nodes. However, for efficient configuration of resources and SLA maintenance, it is crucial that faulty or anomalous behaviour can be detected and accurately localized to a certain link or node in *any* part of the network.

The contribution to the CloNe fault management functionality addresses the above challenges, and includes algorithms for detection of performance degradations, faults and anomalies, as well as fault localization and root-cause analysis. Throughout the project two algorithms [44, 45] have been developed following the functions of the CloNe management as described in [2], implementing functionalities within the scope of Fault Management (FM). The algorithms cover fault management functionality within individual virtual network connections as well as across stacked virtual layers. Moreover, the algorithms are designed for decentralized operation and may be used in both intra- and inter-provider environments, given that relevant topology and measurement information can be accessed in fault monitored resources. Information exchange between fault monitoring nodes and management functions (such as reports about modelled QoS and detected events) as well as algorithm configurations can be done through the management interfaces and the resource administration interfaces [2]. Disseminated information is used for the purposes of re-allocation or updates of resources and goals within RM or GT (e.g. for fault handling), and for information exchange with other FM algorithms.

Virtual overlays and network connections are based on active and passive network equipment, where the endpoint nodes directly measure the QoS across connections of intermediary, passive points, towards other endpoints. With the current use of management and monitoring tools, direct localization of faults, anomalies and performance degradations on individual links between intermediary points is either impossible or complicated in terms of additional measurements (using e.g. external probing nodes). Here, we propose a statistical approach to detect and localize changes in individual link performance based on deriving link delay estimates from end-to-end measurements [45]. Initial results suggest that our approach can successfully model link segment QoS and that changes on individual link segments can be efficiently localized, thereby reducing diagnostic and troubleshooting efforts. As input, the algorithm needs topology information, responses from monitored equipment and algorithm configuration parameters (here detection thresholds and model

parameters for the overlapping estimators). The outputs of the algorithm are estimated QoS parameters (here link delay and packet loss) for individual links on the connection, as well as detected and localized performance degradations.

As virtual overlays depend on shared resources, a scalable and efficient solution is a necessary part of the FM function to effectively pinpoint the true root cause of a service failure detected in one virtual layer, that may be caused in some other, virtual or physical, layer. Based on the management functions, our approach operates in a decentralized manner, and targets the problem of determining the root cause of spatio-temporally relevant events, as described in [44, 2]. The algorithm collects events that are topologically relevant within a certain time period across stacked overlays in the nodes. Furthermore, the algorithm can be triggered on demand by the operator or automatically - either in intervals or when certain detection criteria are met. As input, the algorithm needs topology information and access to generated historic events in monitored nodes. The output is a set of spatio-temporally ordered events used for localizing the root cause or origin of a performance degradation, fault or change in stacked virtual layers. The spatio-temporally ordered events are then handled locally in the triggering node and/or shared (enabled through management resource administration interfaces) with relevant recipients within the management functions (FM, RM or GT), for the purposes of further analysis and for adaptation of monitored services in accordance with QoS requirements and SLAs. As the algorithm can operate over any types of virtual connections or nodes capable of implementing the protocol, the algorithm can be set operating over both intra- and inter-provider environments, given that relevant information can be accessed.

4.4 Implemented Approaches to Resource Management

Cloud networking resource management is a diverse set of management tasks involving both allocating the appropriate resources and configuring those optimally, as well as run-time re-optimization and service migration. This section covers several approaches to allocation and optimization of compute and network resources. Each approach has addressed a particular challenge, or aspect, of resource management. The section also describes an approach toward devising an API for unified management the diversity of resources and rather large number of available configuration strategies.

For network operators, one of the major challenges when providing FNS lies in the efficient embedding of a virtual network (VN) onto a physical network. Since this process requires the simultaneous optimization of virtual nodes and virtual links placement, it is complex in nature and requires large amounts of computing power. Some authors [46, 47, 48, 49] have already proposed solutions to this problem, mostly based on heuristic approaches, but have failed to provide the optimal solution for each VN mapping. This challenge is addressed and the approach is described in Section 4.4.1.

A key problem in resource management is that of mapping a set of applications onto a system of heterogeneous nodes (that provide resources to those applications) and, for each such node, assigning local resources (such as CPU, memory, storage, network bandwidth) for the applications mapped to it. The quality of resource allocation is measured from the point of view of the provider (i.e., how well the allocation adheres to the provider's management objectives, such as load balancing or energy minimization) and from the point of view of the customer (i.e., how well are the goals for each customer application achieved). This quality measure is often made through a utility function whereby an optimal allocation maximizes such a system utility. As the resource demand of customer applications may change over time, the resource allocation process needs to be dynamic, in order to ensure that the system utility is maximized at all times. Optimal resource allocation in the sense of utility maximization is often computationally expensive. This challenge has been addressed by devising a number of heuristic protocols for resource allocation optimization. See

Section 4.4.2.

As different kinds of cloud resources tend to be requested jointly, e.g. compute, storage, and network resources, it is important to enable also joint allocation of the requested resources. An approach to joint allocation of compute and network resources is given in Section 4.4.3.

To cope with high loads, service providers construct data centres with tens of thousands of servers[50]. Such data centres, which often provide a single service, may be located in different places around the world. The actual choice of the specific server to process a given request has a critical impact on the overall performance of the service. Congested servers may introduce service delays, networking bottlenecks around the server, and may further deteriorate the service, due to dropped packets. The server to service-request assignment is a very difficult optimization problem; in many cases, there are multiple objectives and many parameters that should be considered. For example, the current load of available servers, which can be expressed in term of the number of pending jobs, or the estimated time to complete all pending jobs, is an important input parameter. Also network latency can also be taken into account for job scheduling. These and other parameters can be reported, estimated or learned for the server assignment problem. Regardless of the exact optimization criterion, any adaptive system that attempts to address this problem incurs a significant amount of overhead and possible delays, just by collecting the needed parameters from the various network locations. This challenge has been addressed with a scheme called Oblivious Load Balancing. The approach is described in Section 4.4.4.

The issues of efficient resource allocation and utilization have direct impact on business models and the overall success of cloud computing and networking. One way to improve efficiency of resource allocation and utilization in the setting of cloud networking is to enable infrastructure service providers to predict the load and stress on its virtual and physical resources. This challenge has been approached by addressing probabilistic network bandwidth management in the context of Video on Demand (VoD) servers (Section 4.4.5).

Finally, the diversity of resources being jointly requested and the large number of allocation strategies calls for a unified request and management API. An approach to such an API is given in Section 4.4.6.

Demand prediction, as described below in Section 4.4.5, can be used for supporting the resource allocation and optimization functions described in the Sections 4.4.1, 4.4.2, 4.4.3, and 4.4.4. Specific to the resource allocation optimization (RAO) function, 4.4.1 investigates the problem of optimizating network resource allocation and section 4.4.2 investigates scalable optimization of compute resources based on gossip protocols. Section 4.4.4 addresses a the specific allocation problem of load-balancing. Section 4.4.3 investigates joint allocation of compute and network resources.

4.4.1 Network Optimization

Flash Network Slices can be materialised through different kinds of network technologies; one of this technologies is network virtualisation. Network virtualisation can be described as the best fit to provide cloud networking services due to its nature in terms of dynamicity and flexibility.

This section presents a proposal of an integer linear programming (ILP) formulation to solve the Virtual Network (VN) assignment problem and to provide the optimal bound for each VN embedding. The formulation supports heterogeneous virtual and substrate networks. Simulation experiments show significant improvements of the VN acceptance ratio when comparing the ILP formulation with an existing heuristic [49]: in average 10% more of the VN requests are accepted, which corresponds to more efficient use of the physical network. For further details on the proposed mathematical model, please refer to [51].

The problem of embedding VNs is solved by using an ILP formulation [52]. Assume that there are P physical nodes and V virtual nodes. Then two binary assignment variables are used for the

VN mapping process: x for the virtual nodes, where $x_i^m \rightarrow N^V \times N^P$ matrix (equation 4.2)

$$x_i^m = \begin{cases} 1, \text{virtual node } m \text{ is allocated at physical node } i \\ 0, else \end{cases} \tag{4.2}$$

and y for the virtual links represented in equation (4.3), where $y_{ij}^{mn} \rightarrow (N^V)^2 \times (N^P)^2$ matrix.

$$y_{ij}^{mn} = \begin{cases} 1, \text{virtual link } mn \text{ uses physical link } ij \\ 0, else \end{cases} \tag{4.3}$$

The objective function (4.4) is divided into two parts. The first part minimizes the maximum load per physical resource. In the case of having different mapping solutions with the same maximum utilization, the second part is activated opting for the solution that consumes less bandwidth.

$$minimize\ C_{load}^{max} + M_{load}^{max} + B_{load}^{max} + \epsilon \times \sum_{m,n \in N^V(m), n<m} y_{ij}^{mn} \times B_{mn}^V \tag{4.4}$$

where C_{load}^{max}, M_{load}^{max} and B_{load}^{max} are upper bounds of the maximum CPU, memory and bandwidth utilization of the all network (See [51] for further details). In order to solve the problem, further constraints are added, as described in [51]. In short, we need to add constraints for ensuring that each virtual node is assigned to one physical node; that no physical node accommodates more than one virtual node per VN request; that the available capacity of all physical nodes and physical links is not exceeded; the requirement on the CPU frequency is not violated. In order to optimize the mapping of the virtual links and at the same time to cope with the optimization of the virtual nodes, we apply also the multi-commodity flow constraint [53] with a $node - link$ formulation [54], and we also use the notion of direct flows on the virtual links, which is represented in equation (4.5).

$$\forall m, n \in N^V(m), m < n, \forall i : \sum_{j \in N^P(i)} (y_{ij}^{mn} - y_{ji}^{mn}) = x_i^m - x_i^n \tag{4.5}$$

We used several performance metrics to evaluate the optimal model and the heuristic algorithm[49]. We measured the acceptance ratio and the number of accommodated VNs as a function of the number of VN requests per time unit. Details of the simulation results are published in [51].

4.4.2 Scalable Compute Resource Optimization

CloNe addresses the problem of managing compute resources for a large-scale cloud environment, with the objective of serving a dynamic resource demand for various management objectives. While more general, the presentation of the approach focuses here on Infrastructure-as-a-Service (IaaS) where an elastic cloud service provider is hosting VMs in the cloud environment. The VMs are the mechanisms through which compute resources are provisioned in flash slices. The cloud provides *service elasticity* in the sense that it dynamically adapts the number of instances of VMs running a specific application, and the amount of resources allocated to each instance, in response to change in demand of for the application running in the VM. This work contributes to resource adaptation and optimization (RAO) function in the management architecture. The solution approach centers around a decentralized design, where the components of the RAO middleware run on every server of the cloud environment. The approach extends existing management software for private clouds (e.g., OpenNebula, OpenStack, AppScale and Cloud Foundry), by combining and integrating dynamic adaptation of existing resource allocation in response to a change; dynamic scaling of resources for an application beyond a single physical machine; and scalability beyond 100,000 servers.

The solution takes departure in a number of optimization problems corresponding to specific design goals:

1. Performance objective: the allocation of resources should achieve management objectives that are specified by the manager of the cloud at run time while achieving the goals set for each application. In particular, we consider two management objectives: *fair resource allocation* and *minimizing energy consumption*.

2. Adaptability: The resource allocation process must dynamically and efficiently adapt to changes in the demand.

3. Scalability: The resource allocation process must be scalable both in the number of servers in the cloud and the number of VMs the cloud hosts.

The problem of resource allocation optimization is formulated as an optimization problem whose formulation depends on the specific performance objective and on a model for resource allocation, developed as a part of the work in the project. The resource allocation model assumes discrete time, where, e.g., demand changes and resource configurations occur at discrete time steps. Consequently, the optimization problems are also stated with the discrete time assumption. This assumption facilitates subsequent analysis and simulation of the proposed solutions. The RAO problem is formulated as a utility maximization problem with two prioritized objective functions; one for maximizing the utility of a configuration $A(t+1)$ (of all machines in, e.g., a single provider) together with a demand $\omega(t+1)$ on compute resources, in a state $(t+1)$, and the second for minimizing the cost of reconfiguring the machines from the configuration $A(t)$ (in state, or time point, t) to the configuration $A(t+1)$. These two objective functions are optimized with respect to constraints on correct demand splitting, CPU capacity, and memory capacity. For details about the formulations (and solutions) of the optimization problems and the resource allocation model, please refer to [55, 56, 57].

CloNe proposes a generic gossip-based protocol for the RAO problem, which can be instantiated for various management objectives, called GRMP (Generic Resource Management Protocol). GRMP runs on each machine of the cloud that hosts VMs. Two instances of the protocol have been developed and prototyped. The first instance, GRMP-P, is a protocol developed for the objective of fair allocation of resources among the hosted VMs [55]. The second instance, GRMP-Q, allocates resources to hosted VMs such that it uses as few machines as possible, allowing the remaining machines to be put on standby and hence reducing the energy consumption of the cloud [56].

The performance of GRMP-P and GRMP-Q has been evaluated through simulations using a discrete event simulator. In the simulation, each machine in a distributed system executes the protocols in response to a significant change in CPU resource demand. The performance of the protocols was evaluated under varying intensities of CPU and memory loads. The results show that the protocols perform well. Specifically, for GRMP-P (i.e., the fairness protocol), the fairness metric approaches that of an ideal system with decreasing memory load factor (MLF) (i.e., low memory load). For GRMP-Q (i.e., the energy reduction protocol), the fraction of machines freed up is close to the ideal system when MLF is low. The results also show that all key metrics considered in the evaluation do not change with increasing system size, making the resource adaptation approach very scalable, beyond 100,000 servers. Details of the simulation results are published in [55, 56, 57].

An implementation of GRMP that builds upon the OpenStack cloud management platform has been included in the CloNe prototype[3].

4.4.3 Joint Resource Allocation

The two previous sections, 4.4.1 and 4.4.2, address the resource allocation problem for network and compute resources, respectively. Dealing with network and compute resources in a separate way

can be a viable approach in today's world, a world where there is a clear demarcation between network and compute resource management. To be able to meet more complex requests for virtual infrastructures with heterogeneous resources, we need also to consider joint allocation mechanisms. In line with the preliminary work presented in [2], we here propose an algorithm for allocating virtual infrastructures, i.e., a joint resource allocation for compute and network resources. Presented in [58], the proposed algorithm tries to provide solutions to what it is known to be an NP-hard problem.

The algorithm is inspired by the concepts of node and link stress, i.e., links and nodes with less stress are more prone to accepting new virtual resources. Its management objective is to minimize the stress of resources and to balance the stress among resources, so that the physical infrastructure can accommodate as much virtual infrastructures as possible.

Stress is an indicator of how likely a physical resource is to host a virtual resource in comparison with other physical resources. This indicator is used on the mapping algorithm to calculate the potential of a certain physical candidate to host a virtual resource. However, the joint compute and network approach requires that the potential of a candidate is calculated considering the stress of physical nodes (compute or network nodes) and the physical links which might be used to host virtual links.

Algorithm 1 Virtual Infrastructure Allocation Algorithm

1: Calculate link stress;
2: Calculate network and server node stress;
3: Find substrate candidate nodes for each virtual node;
4: Find possible paths between the candidates and a virtual neighbour;
5: Remove candidates without any possible path to one virtual neighbour;
6: If all virtual nodes have a candidate continue, if not stop;
7: Select unmapped node with less candidates;
8: Calculate potential (product of link cost and server stress) of each candidate;
9: Choose the candidate with the highest potential;
10: Save list of candidates and possible paths;
11: Remove non-selected candidates;
12: If there is no candidate left for a given virtual node restore saved data and remove the last chosen candidate from the candidate list;
13: If there are virtual nodes left to map jump to 7;
14: For each virtual link choose the mapping solution with lowest link cost;

Algorithm 1 provides a brief overview of the allocation algorithm. A detailed description of the algorithm can be found in [58] along with simulation and experimental results.

In this section the proposed stress formulas are highlighted. Equation 4.6 presents the link stress formula, which is the value of the bandwidth in use in the physical link. The stress of a network node is given by equation 4.7, and the stress of a computing node is given by equation 4.8. With respect to the network node stress, memory, Central Processing Unit (CPU) load, CPU frequency and the number of active VMs are the parameters taken into account. MEM_{medReq} represents the average memory of the virtual nodes, $Load_{medReq}$ represents the average load increase for each virtual node embedded and k represents a constant value. This way, when calculating the potential of the candidate nodes (which is a product of the link cost and node stress), link cost will be the most important parameter, until the considered nodes achieve a critical occupation state (that can be adjusted through the constant k, which the best values were reached for the value of 3).

$$S_L = BW_{occupied} \tag{4.6}$$

$$S_N = \frac{NumberofActiveVMs}{CPU_{freq}} \left(1 + \frac{k.Load_{medReq}}{N.CPU - Load + \sigma}\right) \left(1 + \frac{k.MEM_{medReq}}{MEM_{free} + \sigma}\right) \tag{4.7}$$

$$S_S = NumberofActiveVMs \left(1 + \frac{K.Load_{medReq}}{N.CPU - Load + \sigma}\right) \left(1 + \frac{k.MEM_{medReq}}{Mem_{free} + \sigma}\right) \left(1 + \frac{k.STG_{medReq}}{STG_{free} + \sigma}\right) \tag{4.8}$$

As to computing node stress formula, memory, storage, CPU load, and the number of active VMs are the parameters taken into account. The formula is similar to the network one, preventing long physical paths from being used when the occupation of the nodes is not at a critical level. Despite not being present in this formula, the CPU frequency is taken into account in the algorithm as an exclusion condition. Note that in the node stress equations σ is a small constant to avoid dividing by 0.

4.4.4 Oblivious Load Balancing

The challenge of load balancing of a large distributed server system (such as a cloud) has been addressed [59] in a way that conforms with the architecture of CloNe. The Resource Management function (RM) is responsible for allocating cloud resources, considering users' request and low-level objectives coming from the Goal Translation function (GT). By implementing our scheme, load balancing with minimal overhead is made available. The GT function might specify the threshold resource load rate at which load balancing should be started. The Resource and Performance Monitoring function (RPM), monitors and reports the current load of the cloud resources, for the RM function to consider the activation of the load balancing scheme.

The load balancing scheme implements an oblivious approach that does not use any state information. In addition to the regular job requests that are assigned to randomly-chosen servers, low-priority replicas are sent to different servers. In some cases, the high-priority copy may arrive at a loaded server, while the low-priority copy is assigned to a lightly-loaded server and thus will complete earlier. Since at each server, high-priority jobs are always served before any low-priority job, the performance of such a system is always at least as good as the basic random assignment technique and, depending on the load, it has the potential of offering considerably improved performance.

We show that when servers can coordinate the removal of redundant copies upon job completion, system performance is improved by a factor of 2-5, even under high-load conditions. A complete overhead free approach without server coordination demonstrates a system performance improvement of 15-50%. A hybrid approach is suggested, demonstrating moderate performance improvements at insignificant overhead. In all cases, jobs arrival is based on a Poisson process, and job length distribution is exponential and heavy-tailed.

As already indicated, our load balancing scheme is launched by the RM function. It could be always active, or launched depending on input from the GT and FM functions. It requires no other input (an oblivious approach), but necessitates a two-priority job queue. It might need coordination among servers, but, otherwise, no output. It addresses the system-wide performance of the cloud infrastructure (server assignment of cloud requests), where our scheme enables better server utilization and shorter job completion time.

Our scheme is clearly useful within an intra-provider; once a job request is sent to one cloud, the job should be assigned to a server in this cloud, and, therefore, no inter-provider load balancing is possible. For global services (such as Google search engines), that own or lease multiple cloud infrastructures on a global basis, inter-provider load balancing is possible, utilizing a management function that is outside of a specific cloud. This is clearly an attractive proposition, but might be too advanced. Load balancing needs to be deployed and show its benefits within the intra-provider, before inter-provider will be exploited. It is expected that the cost (in terms of delay or overhead)

for inter-provider management of our load balancing scheme will be higher than the cost within one provider.

4.4.5 Probabilistic Demand Prediction

CloNe is addressing probabilistic network bandwidth management in the context of Video on Demand (VoD) servers. Properties of user behaviour and workload generating mechanisms are captured through mathematical modelling, using modified Markovian epidemic models. Specific statistical properties of this model permit to derive the distribution of the mean workload of a system over a given time frame. Then, we leverage on these quantities to devise probabilistic management policies that adapt the allocated resources to the current need, dynamically. For more details see [60].

The epidemic model for the VoD system assumes that people watching a video will encourage others (gossip) to do the same (this is the epidemic aspect), and that the demand for the video thus grows through word of mouth. The model distinguishes between a set I of people currently watching the video and who can spread the information about it, and a set R of people, which refers to the past viewers. In contrast to the classical epidemic case, we introduce a memory effect (which is valid for certain cases) in our model, assuming that the R compartment can still propagate the gossip during a certain random latency period. I and R have corresponding stochastic processes $(N_I(t))_{t\geq 0}$ and $(N_R(t))_{t\geq 0}$ representing the time evolution of people watching, respectively having watched the video. The model takes into account $\beta > 0$ as the rate of information dissemination per time unit and $l > 0$ fixing the rate of spontaneous viewers, together with the assumption that the watch time of a video is exponentially distributed with rate γ. Another important consideration of the model is the maximum allowable viewers ($I_{\max}$) at any instant of time (reflecting physical limitations of the system). We also assume the number of past (but spreading rumour) viewers at a given instant to be bounded by a maximum value ($R_{\max}$). To obtain further realism in the model, a Hidden Markov Model (HMM) is added to introduce the so-called "buzz effect": an event signifying that large amounts of information about a particular video is spread during a short period of time. Thus, the HMM has two states associated to a different rate β of information propagation: higher for buzz, and lower for a buzz-free situation.

The model is used for generating workload traces. Such traces are further used for probabilistic resource provisioning through application of the Large Deviation Principle (LDP). Calling $\langle i \rangle_\tau \in \mathbb{R}$ the workload sample mean calculated over a given time period τ, the LDP gives its corresponding statistical distribution $\Pr\{\langle i \rangle_\tau \approx \alpha = \exp\{\tau f(\alpha)\}$, where $f(\alpha)$ is a scale-invariant spectrum that can empirically be estimated from the parameters of the model. We use this spectrum for quantifying the probability of the future mean workload of a system.

For the VoD use case, a VoD service provider wants to determine the reactivity scale at which it needs to reconfigure its resource allocation. This quantity should clearly derive from a good compromise between the level of congestion (or losses) it is ready to undergo, i.e. a tolerable performance degradation, and the price it is willing to pay for a frequent reconfiguration of its infrastructure. Let us then assume that the VoD provider has fixed admissible bounds for these two competing factors, having determined the following quantities:

- $\alpha^* > \alpha_{\text{a.s.}}$: the deviation threshold beyond which is becomes worth (or mandatory) considering to reconfigure the resource allocation. This choice is uniquely determined by a CAPEX performance concern.
- σ^*: an acceptable probability of occurrence of these overflows. This choice is essentially guided by the corresponding OPEX cost.

We moreover assume that the LD spectrum $f(\alpha)$ of the workload process is estimated, either by identifying the parameters of the Markov model used to describe the application, or empirically

from collected traces. Then, the minimum reconfiguration time scale τ^* that yields a dynamic resource provisioning and satisfying to the sought compromise, is simply the solution of the following inequality:

$$\Pr\{\langle i \rangle_\tau \geq \alpha^*\} = \int_{\alpha^*}^{\infty} e^{\tau f(\alpha)}\, d\alpha \geq \sigma^*, \tag{4.9}$$

with $f(\alpha)$ as defined above. From a more general perspective though, we can see this problem as an underdetermined system involving three unknowns (α^*,τ^* and σ^*) and only one relation (4.9). Therefore, and depending on the sought objectives, we can imagine to fix any other two of these variables and to determine the resulting third so that it abides with the same inequality (4.9). We stress on the fact that Equation(4.9) does not provide any analytical solution, rather it is a numerical solution where the operator chooses two parameters and finds the third one according to his need.

4.4.6 Customizable Cloud Resource Management

Current software platforms for the management of cloud infrastructures (*e.g.*, OpenStack [17], Eucalyptus [61], OpenNebula [16]) tend to separate resource management into computing, storage, and network management. As opposed to computing and storage management, which have been extensively explored in the last years, network management in cloud environments is rather in its infancy. Therefore, most cloud management platforms rely on external management systems for more complex, network-layer configuration (*e.g.*, DHCP servers, manual VLAN establishment, NAT or forwarding rules on iptables). In current cloud management systems, customers describe the resource they need in a rather simple way and receive back such resources, somehow provisioned by the cloud. While VXDL improves on the expressiveness of the description language, the mechanisms used to perform the resource allocation will remain closely bundled with the cloud management system in use.

The vision for CloNe points beyond the state of the art as it aims at a tight integration of management of network, compute, and storage resources. Envisioned is a unified API that can be used by cloud providers and cloud customers. This API shall enable site/environment (provider) as well as application (customer) specific algorithms for allocating and continuously re-adjusting resources. An approach for devising such a unified API has been made within the work package, with aim of achieving the following goals:

- Robust networking support for cloud environments through the use of modern networking paradigms (*e.g.*, software defined networks (SDN));
- Support for richer specification of virtual infrastructures, including all kinds of virtual resources (*i.e.*, computing, storage, and network) as well as application-specific requirements (*e.g.*, elasticity rules);
- Flexible resource allocation strategies, with programmable APIs, so that operators can describe and run personalized algorithms for application deployment and optimization.

Figure 4.1 depicts the conceptual building blocks of the architecture of the HyFS Manager. The four major components include the *Cloud Resources* which are controlled and monitored by the *Control Logic*. Different types of resources can be controlled by different Drivers, yet all of them provide a unified API to the *Custom Programs* and the *User Interface*.

The *User Interface* represents the interaction point with the HyFS framework. Through it the specification of the requested resources is transmitted to the system. The initial specification consists of up to three parts. All requests need a *description* of the desired resources. HyFS

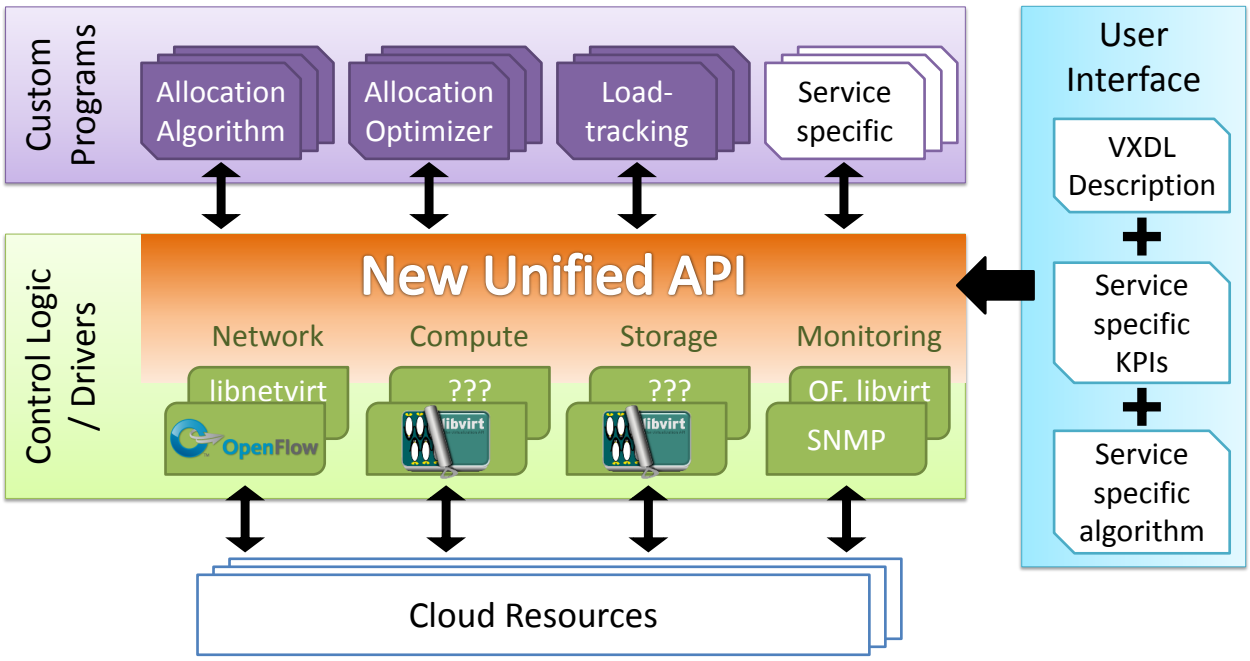


Figure 4.1: Conceptual Architecture of HyFS Manager

specifications can optionally include *service specific key performance indicators (KPIs)* and *service specific resource management algorithms*, extending current cloud control systems functionalities.

The *custom programs* implement the actual management of resources. The programs can either be provided by the user along with a request for cloud resources, or developed by the cloud provider. In practice the cloud provider needs to review the customer provided algorithms before executing them, or the application specific algorithms are co-developed by provider and customer. We envision that a basic set of programs could be shipped together with HyFS, which can then be tuned to the needs of each cloud provider. We envision at least three types of custom programs: *Allocation Algorithms*, *Allocation Optimization* and *Load-Tracking*.

The core concept behind HyFS custom programs is to allow the administrator to plug in and run different algorithms (e. g., centralized vs. distributed, complex multi-objective vs. simple heuristics) depending on the particular needs of the cloud environment. Algorithms may use a set of common functions provided by a programmable unified API implemented by the *Control Logic* to access and modify virtual resource allocations.

The most important part of the *Control Logic* is to provide a resource independent interface for managing the cloud resources. In HyFS this *unified API* allows to manage all of *network*, *compute*, *storage* and *monitoring* on the same level. Pluggable drivers for different types of resources can provide compatibility with a wide variety of existing and future cloud resources.

The concept of providing an allocation algorithm and key performance indicators along with a cloud resource request has also been employed by the *Elastic NetInf Deployment* cross-WP prototype described in Chapter 4 in D.A.9 [62].

A prototype of the unified API was demonstrated at the Student Workshop at NOMS'12 [63] and is described in Section 5.3.3 in Deliverable D.A.9 of SAIL [62]. The cloud management system prototype is called Hybrid Flash Slice (HyFS) Manager, where "hybrid" refers to the integration of network and compute/storage management. At the moment it is designed for a single provider, but allows for multiple data centre locations. Recently, the description of HyFS Manager's architecture

was accepted as a short paper at CNSM'12 [64]. In this deliverable we outline the architecture and refer the reader to the paper [64] for details.

4.5 Implemented Approaches to Security Management

The approach to security management builds on the security requirements and the security goals described in Section 2.6. In the intra-provider case, security requirements and goals are fulfilled through the operation and interaction of five security modules: *SIEM based intrusion detection system*, *auditing and assurance*, *identity management*, *security goal translation*, and *access control*.

Access Control aids entities in the CloNe environment to set and implement access control policies on the underlying resources. Access control policies may either be directly specified by entities with plausible roles, namely, the *infrastructure service user*, *infrastructure service provider* or the *administrator*, or could be indirectly derived from the security goals specified by another entity. The Access Control policy model should cover all types of access control policies and should be easily implementable without large overhead and dependencies on other functions.

The auditing and assurance module maintains records on the fulfillment of deployed performance objectives. It further makes it possible for the participating entities to verify that the security mechanisms functioned properly during a specific interval of time, especially in the case of a security breach. The auditing mechanism should be invoked periodically or on request, and it should be modular, that is, it should compose smaller and independent modules used to audit, assert, and assure specific sections of the overall CloNe infrastructure.

The Identity Management module has three primary features: *identity provisioning*, *authentication and authorisation*, and *compliance*. Identity provisioning and authentication is a critical for enabling efficient management of identity credentials of CloNe entities and virtualized resources. Authorisation is necessary for setting up access control policies depending on the requirements of the CloNe entities. The identity management framework must furthermore comply with the security policies set by the CloNe entities.

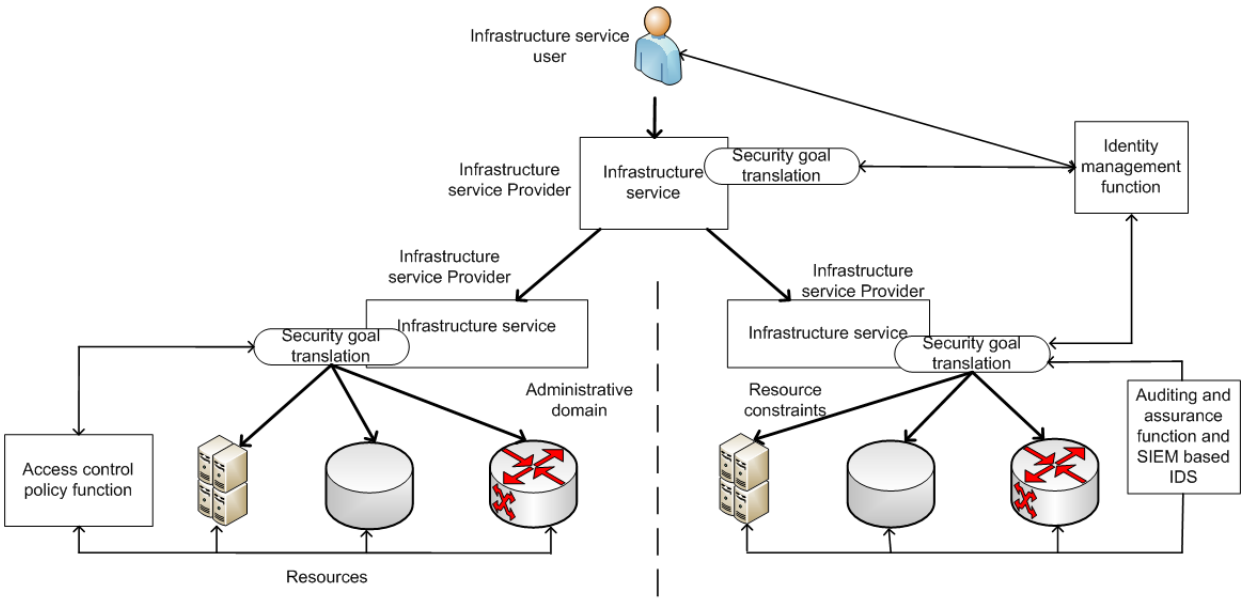


Figure 4.2: Interaction between Security Functions

Figure 4.2 depicts the interaction between the different security functions, namely, SIEM based

Intrusion Detection System (IDS), security goal translation function, auditing and assurance function, identity management function, and access control function. The detailed operations and interactions between these security components are described in Sections 4.5.1, 4.5.2, 4.5.3, 4.5.4, and 4.5.5.

4.5.1 SIEM based Intrusion Detection System

In order to address the security challenges raised by improper misuse protection, a SIEM based IDS [65] has been designed. The SIEM based IDS function detects security incidents which occur in the CloNe infrastructure, and provides a real-time analysis of these security alerts. The SIEM system continuously monitors the underlying infrastructure and security incidents, and these incidents are further categorized and prioritized. This prevents an unnoticed incident to escalate and result in a security threat. Furthermore, the SIEM system also raises flags and security notifications when the risk score of a particular threat exceeds the acceptable limits specified by the security *administrator* or the security goal translation function. The role of the security *administrator* is played by the *Administrator*, *Infrastructure Service Provider*, or both. The security *administrator* determines the acceptable risks, permissible risks, and security policies which are stored in a central policy database. The SIEM system utilizes a policy manager which is responsible for editing security policies in the policy database. Furthermore, the SIEM system interacts with a policy enforcer (Section 4.5.3), which is used for assurance of the implementation of the security policies on the underlying policy database.

An IDS with an acceptable false alarm rate and low operational overhead, which has been customized for the underlying CloNe infrastructure, forms the backbone of the SIEM system [66]. The acceptable range for the false alarm rates of an IDS depend on various factors, for example, the provisioned service, resource availability, involved service users, time of service provisioning, number of parallel users, and the countries where the deployed resources are hosted. The primary role of a SIEM system is to detect and process security events, and take the requisite actions as devised by the security *administrator*. Therefore, it is essential to have an accurate tool to detect and prioritize the security events. The SIEM system uses a new genetic based feature selection algorithm and an existing Fuzzy SVM for effective classification and prioritization of the security incidents. Other IDS models, namely, models proposed by Guo et al. [67], Stein et al. [68], and Cao Li-ying et al. [69] also utilize feature selection and SVMs, and show acceptable performance when implemented solely as a network IDS. However, the proposed IDS has many advantages with respect to the above models. Firstly, it has been customized for the CloNe infrastructure, and integrates seamlessly with the overall architecture and individual functions. Secondly, the IDS has been designed as a host IDS, which is more suitable for the CloNe architecture. Intrusions are detected at individual hosts which reduces the network load and also allows easier correlation between the security incidents generated at multiple host points. Thirdly, intrusions are detected at the network layer, which would allow all network related attacks to be detected and processed. The IDS further utilizes a preprocessing technique which is based on genetic algorithms that can perform attribute selection in an intelligent and efficient manner. Finally, the IDS uses tenfold cross validation, which validates the system decisions and improves the success rate, bringing down the false error rate manifold.

[65] describes the details of the proposed genetic based feature selection approach, and its deployment on a CloNe testbed. The deployment has enabled us to extract features relevant for the CloNe infrastructure from the Knowledge Discovery and Data Mining (KDD) cup data set [70]. These features enable the IDS to classify the base data set in order to detect network-level intrusions at the host level. The process has generated a total of 17 features relevant for the CloNe architecture and have been described in [65]. [65] also compares the detection rates of SVM and fuzzy SVM with feature selection; error rate comparison of neural network (NN), SVM, and fuzzy

SVM; and overall comparison between SVM, fuzzy SVM, and feature selected fuzzy SVM method. The IDS designed for CloNe utilizes feature selected fuzzy SVM and shows lesser error rate and higher detection accuracy. The primary advantage of the backbone IDS is its improvement of the detection accuracy of fuzzy SVM classifiers by extracting and utilizing only the features relevant for the underlying CloNe infrastructure. This improves the overall detection accuracy of the IDS and improves the performance manifold while reducing the overhead on the underlying infrastructure.

Once the security incidents have been detected and processed by the backbone IDS function, they are sent to the SIEM system for further processing. As mentioned in this section, the SIEM system raises a flag and a security notification when the risk score of a particular threat exceeds the acceptable limits specified by the security *administrator* or the security goal translation function. The security threats are the incidents which have been detected and transmitted by the backbone IDS. The SIEM system utilizes an automated engine which computes the threat score of individual threats by utilizing their impact and likelihood of occurrence. These factors have been computed based on sample tests performed on CloNe testbeds, and require more tests for further refinement. The automated engine utilizes attack trees [71] and DREAD [72] in order to compute the threat's business impact and likelihood of occurrence, and its prioritization based on these factors. Furthermore, the engine computes the risk score of each threat using these factors and determines whether the risk score exceeds the acceptable limits. The policy enforcer routinely ensures that the security policies are adhered to, while computing the risk scores and prioritizing threats. The policy enforcer uses the CloudAudit interface [29] and backbone engine. It accepts inputs from the policy manager, the resource management function, and the fault management function. The policy manager inputs the security policies relevant for a particular audit initiated by the policy enforcer, and ensures that the policy enforcer is not overburdened with unnecessary security policies. The policy manager further uses regular expressions to store policies and extract them from the database based on a policy request. The resource management function inputs the current utilization of resources to the policy enforcer, and allows it to run its audits and assurance mechanism when the resource utilization is low. Furthermore, the fault management function provides routine fault reports to the policy enforcer. The policy enforcer in turn checks if those faults were related to security incidents, and filters relevant information to the core IDS for further processing.

4.5.2 Security Goal Translation function

In order to address the security requirements provided by the *infrastructure service user*, the CloNe security architecture utilizes the security goal translation function as its backbone. The security goal translation builds on the goal translation function described in Section 4.2.2, and includes security specific functionalities and utilizes a security analysis engine. The *infrastructure service user* provides its security requirements embedded in the original service request, which are forwarded through the infrastructure service interface and transmitted to the *infrastructure service provider*. The generic *infrastructure service user* request translation process utilizes three processes, namely, service request, optimization, and monitoring, and is described in Section 4.1.

The monitoring process requires close interactions with the auditing and assurance function, and the fault management function, and ensures that the deployed resources adhere to the desired performance and security objectives. Provisioning of security objectives should not lead to performance degradations, and vice versa, which require close interactions between the security and management functions. The auditing and assurance function utilizes a policy enforcer described in Section 4.5.3, which assures the implementation of the security policies on the underlying CloNe infrastructure. These security policies can be appended on the fly with respect to the security requirements specified by the *administrator* and the *infrastructure service user*. If there is a conflict between a security requirement requested by the *infrastructure service user* and a security policy specified by the *administrator*, then the latter is always given preference. In such a scenario,

the monitoring process triggers a re-optimization process, in order to detect whether the affected security policy can be adhered to, by generating alternate provisioning(s) which will not conflict with the given security requirements of the *infrastructure service user*. In case of multiple such instances, a pareto-optimal provisioning is selected. However, if no such provisioning is possible, the *infrastructure service user* is notified about the conflict, and the inability to process his requested security requirements.

The security goal translation function utilizes a security analysis engine, which is a part of the SIEM based IDS and is described in Section 4.5.1. The security analysis engine accepts the security incidents detected by the core IDS and is responsible for raising security threats. These security threats are transmitted to the security goal translation function. The security goal translation function prevents provisioning of any resource which is currently affected by a security incident, and keeps a log of frequently affected resources. These resources are in turn given a lower score by the goal translation function, and are not used for high priority service requests. Moreover, the information regarding faulty resources is transmitted to the auditing and assurance function for in-depth security analysis of the affected resources.

4.5.3 Auditing and Assurance function

In order to address the security challenges raised by isolation and virtualization management, an auditing and assurance function has been designed. It comprises of three sub-functions, namely, a TPM based auditing function, a CloudAudit [29] based assurance function, and a policy enforcer.

The **TPM based auditing function** attests the geographic location of the physical resources provisioned to the *infrastructure service user*. For example, when an *infrastructure service user* specifies a desired geographic location, where its resources need to be hosted, the location constraint is translated by the security goal translation function and forwarded to the auditing and assurance function. The auditing and assurance function invokes the TPM based auditing function [3], which then verifies the actual location of the deployed resources with the desired geographic location given by the *infrastructure service user*.

The **CloudAudit based assurance function** is responsible for verifying that the deployed provisioning match the given security requirements. Moreover, any of the CloNe entities, namely, the *infrastructure service user*, *infrastructure service provider*, and the *administrator* can invoke the assurance function and verify whether the security mechanisms functioned properly during a specific interval of time. The assurance function is especially critical of any security breaches which are detected by the core IDS deployed by the SIEM based IDS. The security breaches are logged along with any additional information that is received from the security goal translation function. The security goal translation function transmits faulty resource information, which is then correlated with the results of the SIEM based IDS, and its own findings. The assurance function builds on top of the CloudAudit assurance interface [29], customized for the CloNe infrastructure. The CloudAudit interface includes a core framework [73] for management and governance of the cloud architecture, which enables the entities to manage and verify the compliance of the deployed architecture with respect to the given policies. Moreover, the assurance mechanism also comprises of a risk management function, which allows risk analysis of *infrastructure service providers* and *administrators*, risk assessment of provisioned resources, management of resources, and generates resulting security policies with respect to the given security requirements. The risk management function supports the security analysis engine which is a part of the SIEM based intrusion detection engine. The security analysis engine is also responsible for accepting security incidents and raising security threats after computing individual risk scores for all security threats. The security analysis engine accepts the inputs from the risk management function, and correlates its results with those of the risk management function. Any discrepancies result in a detailed re-computation of the risk scores. The resulting security policies are transmitted to the policy manager utilized by the SIEM

based IDS. The policy manager matches these given security policies, with its generated security policies and checks for any discrepancies. Any discrepancy results in a detailed security policy regeneration process based on the given security requirements of the *infrastructure service user*.

The **policy enforcer** is responsible for ensuring that the security policies, set by the security *administrator*, are adhered to by the provisioned resources. The policy enforcer maintains a central Extensible Markup Language (XML) file which stores the security policies, and uses Atom [74] for maintaining the policies as individual feeds. Each security policy is composed of a number of components, for example, the issuer, the permitted roles which can access resources, the permitted resources, the access permissions, and exceptions. Each component is implemented as an entry and has an extensible set of metadata that can be attached to it for further extensibility of security policies.

4.5.4 Identity Management

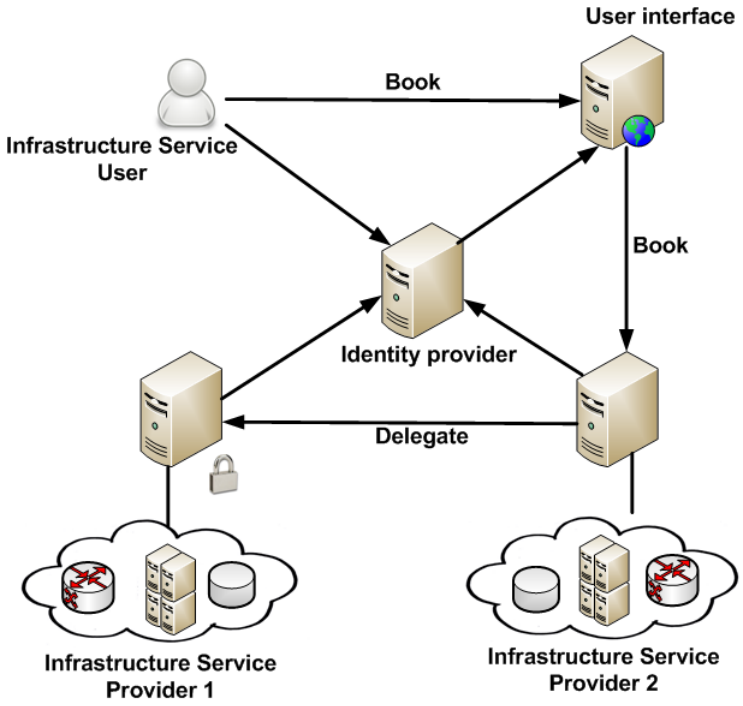


Figure 4.3: Identity provisioning using a central Identity provider

An identity management framework has been developed for CloNe, addressing the requirement of identity management as described in Section 2.6. It is capable of identity provisioning, creating and maintaining access control policies, and authentication of legitimate *infrastructure service users*.

Identity provisioning acts as an integral part of the identity management function, as it aids in the overall objectives of the framework, namely, authentication, authorisation, and compliance. Identity provisioning in the CloNe environment enables provisioning (creation and maintenance) and de-provisioning (removal) of user accounts (user identity, user role). In order to achieve automated provisioning and de-provisioning of user accounts, native SPML adapters are employed at the *infrastructure service provider* side. In a typical provisioning scenario, the *infrastructure service provider* identifies that an *infrastructure service user* requires access to a particular resource. The *infrastructure service provider* contacts the *administrator*, which then performs an SPML based transaction to create a user account for the requesting *infrastructure service user*.

Upon successful creation of the user account, the *administrator* returns an SPML success response to the *infrastructure service provider*. In a typical de-provisioning scenario, the access rights of an *infrastructure service user* to a particular resource or resource set is removed and the relevant account information is deleted by the *administrator*. The authentication of service users and their respective access rights for the underlying virtual resources are handled by the access control policy function (Section 4.5.5). The compliance of the identity management framework is managed by the auditing and assurance function (Section 4.5.3).

The CloNe use cases [4] and architecture (Chapter 2) warrant the need for an enterprise-centric, and not a user-centric identity management function. The resource delegations are handled by the *infrastructure service provider* and the *administrator*, and are kept transparent from the *infrastructure service user*. This results in information hiding (both between the *infrastructure service providers*, and between *infrastructure service providers* and *infrastructure service users*), and also hides the entire resource delegation chain from the *infrastructure service user*. This in turn promotes privacy and anonymization of the participating entities.

Figure 4.3 depicts the identity provisioning by using a central identity provider, which enables user-centric provisioning. A central identity provider, similar to the one use used by User Managed Access (UMA) [75] and OAuth [76] needs to store information on identities, and credentials of all the involved parties. Moreover, it allows unique identity provisioning within the CloNe ecosystem. However, all parties must trust the central identity provider, as it results in a single point of failure.

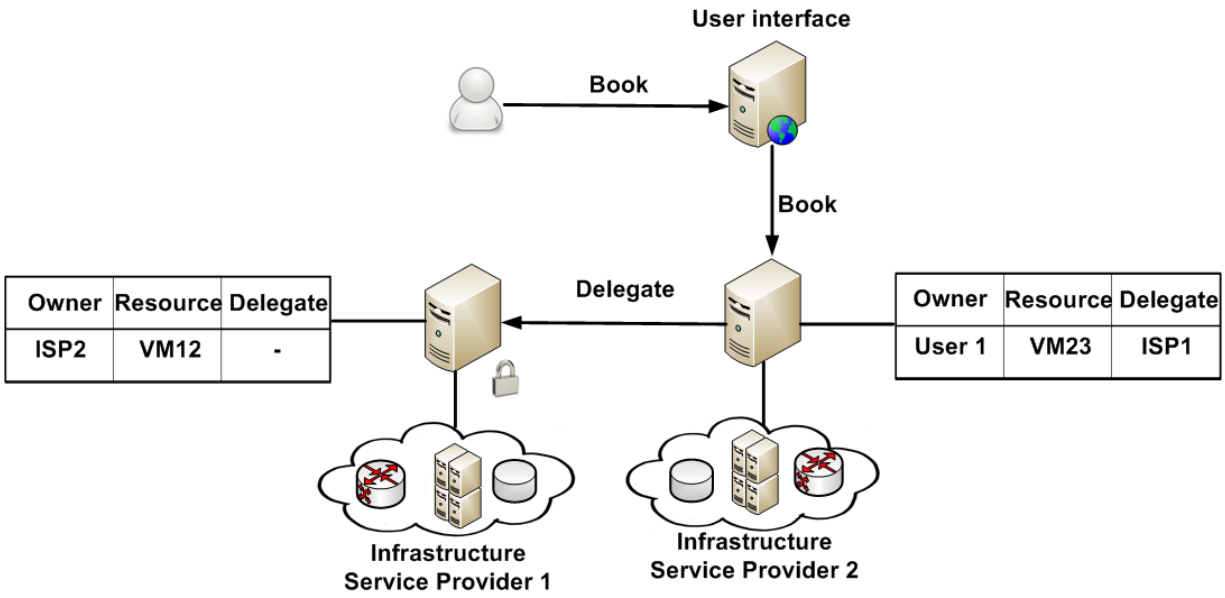


Figure 4.4: Domain-wise Identity provisioning

Figure 4.4 depicts provider-wise identity provisioning with resource delegation between multiple providers. In such a scenario, each *infrastructure service provider* manages identities and credentials of its *infrastructure service users*, and ensures a unique identity for each participating entity in their respective provider. The primary advantage is the lack of a single point of failure, but it requires a trust relation between providers. The CloNe identity management solution and the core access control function utilize provider-wise identity provisioning due to the lack of a single point of failure, and its seamless integration with the existing CloNe architecture. Table 4.2 compares the identity management mechanism used in CloNe with other user-centric mechanisms, namely, OAuth and UMA.

Table 4.2: Comparison between different identity management mechanisms

CloNe identity management mechanism	User-centric identity management mechanism
No external identity provider • Prevents service provider lock-in • Causes additional load on the *infrastructure service provider* for authorisation	External identity provider (IdP).
Low complexity due to simple policy logic/structure and clean security architecture.	Low complexity due to inherent design.
Support for nested groups of *infrastructure service users* and resources without overcomplicating security policies.	Nested groups and role hierarchy realization is hard to implement due to its design.
Authorisation discovery, especially for cases involving multi-level delegation is easier as the authorisation manager employs an authorisation prover to determine authorisations with a subject.	Authorisation discovery becomes tricky, especially in cases involving multi-level delegation.
Supports both user-centric and enterprise-centric identity management and allows easy switching between the two.	Only supports user-centric identity management.

4.5.5 Access Control

In order to address the security challenges raised by information security and virtualization management, CloNe has developed an access control policy function. The access control policy function forms the backbone of the identity management function, and is responsible for determining whether an *infrastructure service user* is allowed access to a particular resource. The access control policy function utilizes an authorisation logic [77] which enables fine-grained control of virtual resources managed by the *administrator*. The authorisation logic manages access policies, which are defined by the respective resource owners. The authorisation logic of the access control policy function utilizes an authorisation prover, which ensures the sanity of access control grants and prevents the circular reference error from occurring (circular reference is a series of references where the last object references the first, resulting in a closed loop). Furthermore, it executes authorisation proofs by utilizing access policies, which validate the user access on a specific resource. The authorisation proof can involve *infrastructure service providers* and *administrators* spanning a single, or multiple administrative domain. The current section will cover the authorisation logic and the core implementation details of the access control policy function.

The authorisation logic enables fine-grained control of the virtual resources which are managed by the *administrator*. The authorisation check of an *infrastructure service user's* access to a resource involves chaining together individual virtual infrastructure delegations and access right delegations. Each *administrator, infrastructure service user,* and *infrastructure service provider* has the ability to delegate the resources under its control to any other participating entity. The authorisation logic is used to encode individual resource delegations as authorisation grants, and chains them together to verify whether a party is authorized to access a resource. Any party that is responsible

for specifying grants is termed as an *issuer*. There are two types of permissible grants in the authorisation logic, namely, authorisations and name definitions. Authorisations cover all resource delegations which occur from an issuer to an issuing party. For example, an *Infrastructure service provider* **'AISEC'** might authorize an *infrastructure service user* **'A1'** to use the resource **'alpha'**. Moreover, a name definition grant specifies that according to the issuer, a name is defined as another name or a user. For example, **AISEC grants : alpha = foo@aisec.fraunhofer.de**, implying that **AISEC** defines the name **alpha** as **foo@aisec.fraunhofer.de**. Furthermore, names can be multiply defined, which allows them to represent groups. The details of the syntax and extent of these grants are described in [3].

The access control policy function utilizes the authorisation server that receives grants from issuers and can perform authorisation proofs. In addition to authorisation proofs, the authorisation server uses its inherent logic to control access to its services, for example the privilege to grant access to the authorisation server. Furthermore, the authorisation server implements a global authorisation service, which can span multiple administrative domains. The details of the authorisation server is given in [3].

5 Elaboration of the Inter-Provider and Service Layers

5.1 Main Concepts

5.1.1 Declarative vs Procedural Knowledge

Procedural programming builds on the idea of procedure calls, which are invocations of a series of computational steps that need to be completed in a given specific sequence that defined the program. In logic programming a program is a set of premises and computation is executed to prove candidate theorems trying to determine on what the problem is, rather than on how to solve it. Prolog-like logic programming languages are executed as goal-reduction procedures. Therefore clauses like $A : -B1, ..., Bn.$ have a dual interpretation, both as procedures (to solve A, solve B1 ... Bn) and as logical implications: $B1 and ... Bn$ implies A. Therefore, declarative programming serves two purposes: making sure that programs are effective and efficient (procedural) and ensure that programs are correct (declarative).

Model-driven development (MDD) is a methodology in software engineering that focuses on the creation and the processing of models. Models aim at a high level of abstraction. Therefore, models often have a rather declarative and a less imperative character. The model is the contextualised into a Domain Specific Language and expressed in a concrete language, which tends to be more imperative in nature than the Model.

From a bird's eye viewpoint, each provider can be conceived as a peer that exchanges information with other peers so that these enact "whatever is needed" for that thing to happen. In other words, they communicate their peers what needs to be done, not the low level specifics on "how to make it happen". Thus, MDD or Procedural programming approaches are taken into our architectural design, where we use a high level declarative language so that the providers can know what is required from them in a higher abstraction level and derive from there how that is actually done in their specific context (e.g. concrete set of technologies being employed). This is exemplified in Figure 5.1, where the peer entity representing provider 1 in the federation receives a high level Model specification indicating what is required (but not how it is going to be done). This is resolved by the provider by applying any internal representation format at the preferred abstraction level (which may be technology dependent). This intermediate internal representation is finally enacted by issuing a series of commands that indicate "how" the request is actually enacted. This realisation may imply that it cannot be done locally and needs to be delegated into another peer.

5.1.2 Distributed Computing Model in CloNe

Centralised models for system management present several inconveniences that make them poorly appropriate for large-scale infrastructures spread across different organisational and geographical domains. In our specific setting a variety of resources (e.g. a block in a storage system) may need to be handled. These elements can leave, join, and fail in a dynamic manner. This large scale and fine granularity of elements to be controlled is nearly impossible to manage centrally and does not present failure-tolerance. In the light of these requirements, the presence of an inter-provider coordination/broker/orchestrator is not feasible. Therefore, we were motivated by the nature of

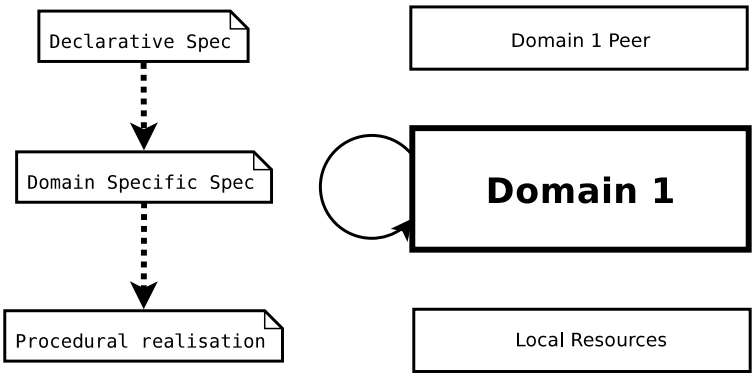


Figure 5.1: Concretisation of the high level declarative model specification into procedural domain and technology specifications

our requirements to adopt an efficient decentralised solution that can dynamically adapt to ever changing conditions of a really large number of objects/resources at a fine grained level (e.g. just picture the huge number of virtual NIC in the VMs of all the data centres in the federation).

In decentralised organization a fundamental challenge in creating and managing a globally decentralised inter-provider environment is keeping connectivity between various untrusted components that are capable of self-organization while remaining fault tolerant. While wide-area scalable overlay of distributed clouds have recently been proposed [78], they still do not deal with the huge problem of creating a heterogeneous inter-provider infrastructure where cloud providers and ISPs can interact with each other. However, the main architectural principles or distributed computing for large scalability remain the same.

This is why we also took a peer-to-peer approach to build all the elements for inter-provider communication, coordination and request realisation. Take into account that every peer entity represented in Figure 5.2 is actually performing several roles at the same time: it is the access point for requests from other providers and the source of requests for other providers when something cannot be done with the resources at hand locally. These requests include handling joining of new peers (providers), deal with capacity advertisements and, of course, requests infrastructure services from other providers (e.g. VM deployment or endpoint connection). Note that in Figure 5.2 a full-mesh network has not been created on purpose to reflect the variety of business relationships held by the participants in the inter-provider scenario.

5.1.3 The Distributed Control Plane

Having a general idea on the specifics of the computing model assumed by our system, we present the main functions that make the system workable, scalable and failure-tolerant, while coordinated. We refer to this set of functions as our "Distributed Control Plane".

5.1.3.1 Provider Federation Formation and Capability Discovery

Peer to peer literature is abundant in how peers join and depart from the federation. When attempting to join a P2P network, specific bootstrapping or rendezvous protocols may be required to communicate with other nodes and finally join the network. Furthermore, to add complexity to the global picture, these protocols and configuration requirements may dynamically change as the infrastructure and membership of the P2P network evolves. Therefore, there is a need to be able to dynamically inform a newly joining node of the required protocols and configurations.

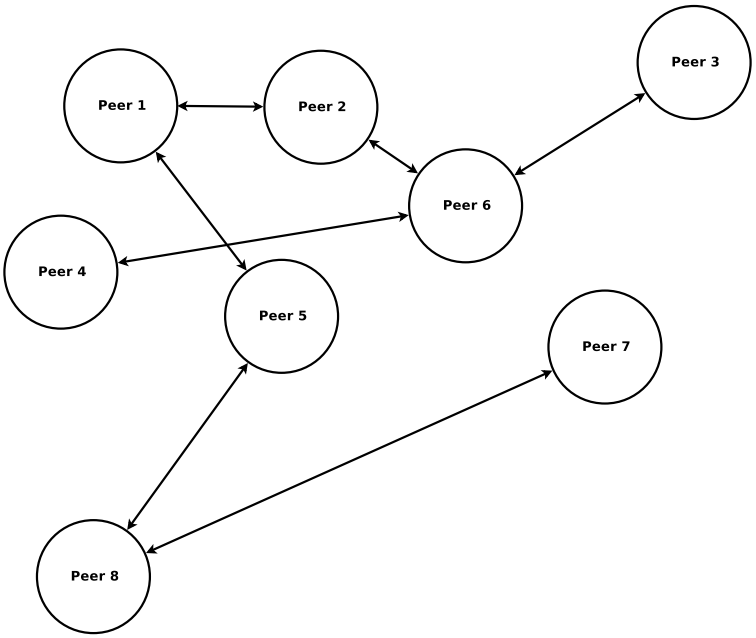


Figure 5.2: Bird's eye view of the system employed for inter-provider coordination.

In our high level specification, we will assume here the simplest scenario, which is that any given provider (peer) will join by means of connecting to another known peer (provider) in the system. This situation reflects the reality of ISPs and data centre operators, which are inherently joined by means of their actual networking constraints. The key element exchanged by joining providers is that of its real capacities (e.g. large scale compute power; or WAN connectivity with guaranteed QoS between country A and B) specified in a high level language described by a shared inter-provider model all the providers need to understand.

Upon receiving this capability description, the receiving provider may adopt two different strategies: 1) relay the advertisement of that capacity and 2) integrate that capacity into its own offer (adding a profitable margin) and re-sending its capacity to its neighbour as an update. This latter strategy may be convenient if the provider wants to outsource part of its capacity (hybrid cloud). The former strategy is more appropriate, for instance, in the case of a network operator connecting several data centres. Since the network operator is not directly competing with the data centres, it is in its best interest to advertise as many data centres as possible so that its network can be used as a link between any two of them (thus increasing its benefits).

5.1.3.2 Delegation of Implementation

Delegation of implementation is a service layer activity. A given provider may decide not to implement a client's request itself, but delegate part of the (or the whole) request to another provider. This fact may be hidden from the requester at the provider's discretion. Following the declarative vs. procedural discussion above, the idea of delegating responsibility on the final enforcement of "what needs to be done" is inherent to our architecture. Delegation follows naturally. If a provider cannot fulfil "what" is asked or it does not have the have the means to resolve "how" to do it (e.g. does not use a given protocol or hypervisor).

5.1.3.3 Name (Reference) Resolution

A naming scheme is a key component of any distributed system. Names (references) are mapped to objects or their locations and vice versa using a name resolution service. Probably, the most well known example of such a system would be the Domain Name Service (DNS). Unfortunately, DNS is not enough for handling resources at a fine granularity level (e.g. a VM NIC) in a highly dynamic environment where the resources pop up and disappear in minutes. In general a name or reference may have the same Common Internet Scheme Syntax of a URL (in the 1738 RFC sense). This naming mechanism may be modified by the specific needs of the provider. When a resource is delegated to an external provider, the other provider may be using a different scheme that is not compatible and, therefore, a mapping is needed.

When the resources need to be accessed (e.g. when a link needs to be negotiated) these abstract references are resolved in order to find the actual location of the resource. The resolution of the references may occur in two differentiated manners:

- There is a DNS like hierarchy that resolves the abstract name into a concrete one. This approach presents the typical limitations of a centralised service and cannot cope with the dynamism required in large settings such as the one depicted by CloNe.
- The resolving provider follows the references (think of each reference on each provider as a pointer in C programming) and asks all the providers in the delegation chain until the one that finally instantiated the resource responds with the concrete (routable) locator. This approach has the advantage of letting providers use a wide variety of schemes of their own, with no need for any synchronisation of neither naming nor global entities. As a down size, this mechanism requires following an undetermined number of hops until the final "pointer" is found and the reference can be materialised.
- Pure peer to peer naming mechanisms that similar to the distributed hash table employed by systems such as Chord or Pastry. In this inter-provider setting, every object in every provider will use an *ad hoc* naming mechanism and this local name will be hashed to produce a N-bit peer identifier. The DHT is then employed to retrieve the location of the host publishing that identifier, the host where the required resource can be found. There are however some significant differences when compared to Chord or Pastry. These systems store the indexes of objects (hashes) at the node whose identifier is closest to the hash, and the routing algorithm is designed to find that node. In contrast, in our approach the hash is stored on the node that publishes the identifier. A node will thus have as many entries in the routing system as the number of identifiers that it publishes. Most DHT systems assume that only one node publishes a specific index. In contrast, in our multi-provider scenario collisions are indeed possible and multiple hosts could publish the same name. The internal index is in fact composed of the N-bit hash of the peer (provider) name and a N-bit (resource) location identifier, which can be derived from an IP address of the publishing node. Therefore, using DHT techniques for naming and discovery of the location of resources (routing) prevents from relying on a centralised (brokered or orchestrated) architecture that would pretty soon prove poorly scalable in the case many resources were to be (un)published over relatively short periods of time. Also, this prevents the system from having a single point of failure that could cause a massive denial of service (resources not being found) with huge economical implications for the service providers and their clients as well.

5.1.3.4 Message Delivery

While the proposed naming mechanism solves the "routing" of the messages and help to locate the "Object resource", an actual messaging system is required. Knowing the name of the required

object (e.g. VPN end point in VPN server) is enough to locate it and invoke its exposed services. Once the name(reference) to network endpoint (e.g. protocol, IP and port) mapping has been resolved, one has to take into account that most of the messages will be sent in an end to end manner, although the message may be diverted upon reaching an intermediate destination, due to a possible delegation.

Other possible configurations are needed so that an object can be subscribed to updates in the status of any other object located anywhere in the provider federation. Therefore, an asynchronous delivery mechanism is obviously preferred to cope with the required scale of the system. One should be aware that several objects may be interested in the status (or updates) another object may issue (e.g. VMs processing the same data blocks in a read-only manner may want to be updates on failure of the volume that contains the data blocks; or VMs communicating with each other via a WAN link may want to be notified if the link fails).

In our setting each of the peers (providers) will present an element of the message service that is capable of delivering messages to local resources or re-route a message addressed to an unknown recipient.

5.1.4 Goal Translation

The entity that is in charge of understanding the logic specified in the high level declarative language is the goal translator. This is one of the main inter-provider functions. Its basic elements are similar to the ones for the intra-provider case detailed in Section 4. However, an additional layer of complexity is added here: there are several providers in the picture. Therefore, one of its main building blocks is a domain decomposer: an entity capable of handling a more procedural language only, analysing the logic given by a new declarative request is required and making appropriate decisions on exactly where different pieces should go.

This component may need to get the names referred to into the specification and resolve them into something more meaningful, according to provider specific contextual information (e.g. a given private range IP). The interpreter is also in charge of determining whether or not the specified elements exist already (either in that provider or in a delegated from in another remote provider). If the elements exist, the interpreter will make inferences on the actions that are actually needed in order to reach the desired state (e.g. add a new data block to a previously existent VM or connect it to a new virtual network in a remote provider). Once these inferences have been made, the interpreter is in a right spot to start generating a more procedural format that the decomposer can understand and map into two possible outcomes:

- a series of local calls in the provider-specific technologies (e.g. eucalyptus-type EC2 calls to deploy a VM). This implies requesting actions from the local infrastructure services.
- a delegation request (just a service call in the declarative form) into a remote provider deemed to be more appropriate for the whole (or part of the) request. This implies invoking the remote APIs exposed by the selected provider.

In both cases, the interpreter is also in charge of handling messaging actors instantiated for the requested elements (objects) so that remote or local elements can subscribe for updates on their status. See Section 5.3.1 for a concrete implementation of this function.

5.1.4.1 Security Functions

This entity is responsible for ensuring that the virtual resources can only be accessed according to the security policies specified by the CloNe entities is the access control function. The Goal Translator may generate either a local call in provider-specific technologies, or a delegation request

into a remote provider that is deemed to be more appropriate for the entire, or a part of the resource provisioning. The latter is relevant for the inter-provider scenario, and requires the access control function to manage the access to the resources with respect to the individual virtual infrastructure delegations and access right delegations. The details of the working of the access control function in the inter-provider scenario are described in Section 5.2.3.

Furthermore, all the supporting security functions, namely, SIEM based IDS, security goal translation, auditing and assurance, and identity management execute solely in the intra-provider scenario, and ensure that the incoming remote request do not violate any security policies defined for their respective provider.

5.2 Implemented Approaches to Inter-Provider Management and Security

In this section we will present the most relevant elements of the aforementioned architecture that have actually been implemented and demonstrated in or prototype. These have been selected since they clearly illustrate some of the most required elements in inter-provider scenarios that place CloNe beyond the state of the art. We also highlight the main limitations that will guide future development of CloNe architectures and prototypes.

5.2.1 Object Location

As an example of message service, CloNe has implemented a message exchange framework called Cloud Message Brokering Service (CMBS). From a technical perspective, CMBS is a solution for exchanging messages between cloud providers and especially cloud networking providers.

CMBS is based on an Event Driven Architecture (EDA) where the events are given in the form of messages. In such message-oriented system there are three distinct layers that could be used: application layer (e.g. Java Message Service (JMS)), transport protocol wire layer with a broker (e.g. Advanced Message Queuing Protocol (AMQP)) and a transport protocol wire layer without a broker (e.g. ZeroMQ). In an attempt to reduce the number of single points of failure and to improve the scalability of the system CloNe's CMBS has fully relied on ZeroMQ for message exchange.

Exchanging information between cloud providers can have several patterns and can occur in different ways like sending information to one or many providers, or sending information related to a specific topic. To satisfy the Cloud networking requirements of the information exchange, CMBS is based on a message exchange pattern that is structured by five layers:

- *layer 1* - cloud provider discovery space: The role of this layer is simply to exchange information about the type and characteristics of cloud providers. Each cloud provider who wants to be a member of the CloNe DCP should advertise about him self through this layer. This function is implementing part of the peer-to-peer discovery protocol outlined above.
- *layer 2* - broadcast space: enables sending information to all known members of the DCP.
- *layer 3.1* - unicast space: via this layer, a provider can send information to a particular list of providers $\{X_1, X_2, ..., X_n\}$.
- *layer 3.2* - interest cast: allows providers to send data about a list of topics $\{Y_1, Y_2, ..., Y_n\}$ to all providers that are subscribed to this topic.
- *layer 4* - uni interest cast: via this layer, a provider can send information about a list of topics $\{Y_1, Y_2, ..., Y_n\}$ to a particular list of providers $\{X_1, X_2, ..., X_n\}$.

Thus, by relying on proper message specification, we have built a decentralised architecture for message exchange in CloNe. More details are presented in the deliverable D-5.3 Description of the implemented prototype.

5.2.2 Link Negotiation Protocol

A virtual infrastructure specified in tenant request can span multiple providers, and in such cases the individual "pieces" of this virtual infrastructure within each provider's administrative domain needs to connect to one or more distributed "pieces" in the other providers' administrative domains. There are several logical and physical entities that need to work together in order to establish the connectivity that spans multiple providers. The actual connectivity between administrative domains needs more detailed information such interface addresses and agreement on routing protocols which are not available in the original request exposed by the decomposer. A specific domain-to-domain communication mechanism is required so as to enforce the actual communication.

Naturally, the link may cross provider boundaries, and therefore will make these connections. For this purpose a *Link Negotiation Protocol (LNP)* was defined. This protocol is responsible for creating one or more virtual links belonging to the same virtual infrastructure but may be spanning multiple (usually two) providers' administrative domains.

The protocol achieves the following high-level objectives:

- Simple and low level protocol agnostic
- Support of various transport network solutions (L2, L3)
- Agnostic to any particular networking implementation

After the completion of the protocol, bindings at three different levels are achieved between the two infrastructure providers with respect to the link, namely

- Agreement on the link reference at the virtual infrastructure level.
- Agreement on a data transmission link for sending data from one domain to another. This are already installed and configured by the service provider and include physical layer separation schemes like Provider Backbone Bridging or MAC-in-MAC, or tunnelling schemes like IPSec or GRE.
- Agreement on a logical link part spanning between two domains. This corresponds to virtual link in the virtual infrastructure of the tenant and has a one to one mapping. The difference is that a virtual link is unique only to the virtual infrastructure, whereas this logical link is unique across the two domains in the negotiation process. For example VLANs may be used for creating the logical link in which case the VLAN number is used for uniquely identifying the logical link. The additional configuration parameters, such as interface addresses of the two endpoints and a routing protocol in the case of an L3 link are negotiated on this logical link.

LNP facilitates the creation, update and deletion of the logical links. More details of the link negotiation protocol, including the various operations, parameters and its implementation in data centres and WANs are described in detail in deliverable D-5.3 Description of Implemented Prototype [3].

5.2.3 Access Control

The access control function is capable of executing in an inter-provider setting, and is responsible for an *infrastructure service provider* to delegate the implementation of its resources to other *infrastructure service providers*. Moreover, an *infrastructure service user* could himself be acting as a provider to other *infrastructure service users*, or *infrastructure service providers*. The access control function ensures that the delegation request that has been transmitted into a remote provider conforms to the respective security policies of each affected provider, and the individual virtual infrastructure delegations and access right delegations.

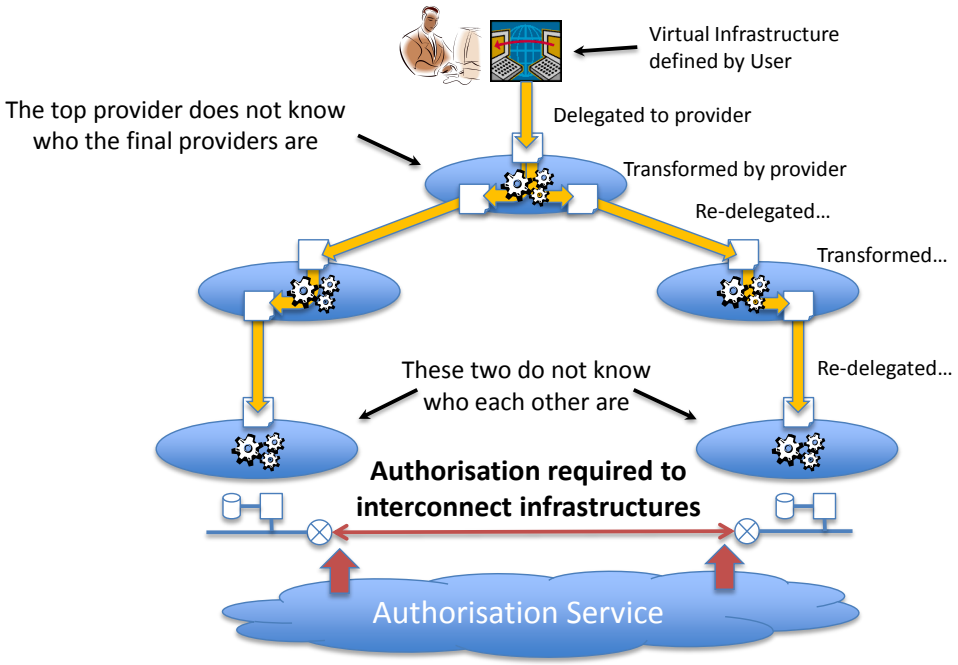


Figure 5.3: Inter-provider infrastructure provisioning

Figure 5.3 depicts an inter-provider scenario, whereby an *infrastructure service provider* has delegated a virtual infrastructure amongst two *infrastructure service providers*. Each *infrastructure service provider* further re-delegates their corresponding part. The *infrastructure service providers* need to establish a link between their respective access points, in order to connect their infrastructures. This mechanism is described in [3].

5.3 Implemented Approaches to Service Layer Management and Security

The service layer is largely based on existing cloud computing software stacks and interfaces. The additional management and security functions at this layer are embodied in the goal translation function.

5.3.1 Goal Translator

During its normal functioning, the goal translator module will end up invoking a series of functions by the execution of procedural commands. This subsection describes one of the most relevant ones:

the decomposer in the light of inter-provider scenarios.

Decomposer The goal of the Decomposer module is to convert abstract or high level requirements into concrete actions to be performed on physical resources. Using a virtual infrastructure high-level description (a VXDL file) as an input, this module makes the translation from the VXDL request to a set of nodes and link requirements and allocates them to the available physical resources. It makes the allocation by sending messages on the providers' interfaces. Providers can be data centre operators or network operators.

The different providers (data centre operators or network operators) are registered on CloudWeaver and specify some of their physical specificities (for example, their geographic position). The VXDL request is split according to the requirements on the physical resources and configuration messages are sent to the providers matching the requirements.

The allocation process, inside the decomposer, maps a graph of virtual resources on a graph of declared network and data centre resources.

As an individual provider might not have enough resources to implement the user request or the user request might specify the use of different providers, it is necessary to be able to delegate the deployment to multiple providers.

To provide this feature, the decomposer can automatically generate the appropriate (intermediate) requests to network providers to instantiate one (or several) FNS (see Section 2.3.1 for a reminder). Once they are requested and later created via the infrastructure service interface, communication will be possible between the different data centre providers involved in the user-request.

5.4 Load-Adaptive Deployment

Our leading vision is to optimise geographically distributed application deployment, so that QoEs like application responsiveness are improved. In using a poorly responding application the user is less efficient in his work, while waiting for the application's response to his interactions. A significant part of the delay experienced is the round trip time for a request between the user's computer and the application's instance or server in a cloud. This round trip time can be reduced by answering requests closer to the user. Only such a geographically distributed deployment will reduce the round trip time and improve the responsiveness.

The deployment close to users will also reduce the link congestions in central networks. This argument is borrowed from Content Distribution Networks (CDNs), where caches are located at the edge routers of the network. With cloud computing, we can allocate resources for application or caches at different locations.

In the near future we foresee a huge amount of potential cloud locations and data centres in which our application can be deployed. We face two issues in order to exploit the benefits of a geographically distributed deployment. At first, we need to select those appropriate locations, which for instance are close to the users. Secondly, we need to be able to deploy our application in an elastic and distributed way.

A dynamic selection of appropriate locations is critical to the QoEs and to cost related to the deployment of the application. User demand usually fluctuates within a day, when people arrive at work or home. When users start to use an application deployed in the cloud, load needs to be controlled and the appropriate service needs to be expanded to deliver the best service to customers.

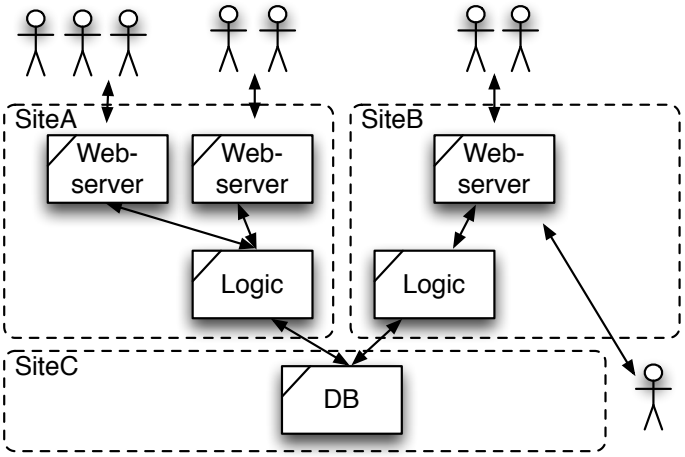


Figure 5.4: Example of a scaled geographical distributed three tier application.

In traditional centralised deployments, most of these effects go down in the overall average usage. But in a geographically distributed deployment, these effects occur at different sites at different times. In a static deployment, we can only predict but do not know the exact usage pattern in the future. As a result, we will most probably suffer from a bad fit of currently allocated and needed resources at different locations. Either we have not enough resources resulting in low QoE or we have too many resources resulting in a waste of resources.

How can we ensure a good QoE without the high cost of over-provisioning? The only way is to enssure there is a dynamic adaptation of the application deployment according to the current usage pattern. CloNe enables parts of the application to be deployed in an elastic inter-provider cloud consisting also on network locations more central than the classic edge data centres. If at some point, some users do not need the application anymore, then some of the parts of the application (either in the network or in the edge data centres) may be removed.

5.4.1 Need for Dynamic allocation and Load Balancing

Typical applications are compositions for distributed components. As an example, Figure 5.4 shows a typical three tier web-application, using a web-server with a logic-server and a central database. The arrows indicate which components are communicating with each other. Users are served by the web-server. The lower right user illustrates a special case: He is not served close by. An extra web-server for him at SiteC is too expensive.

Such a system implements two separated capabilities:

- The *Elasticity* describes the capability of a Virtual Infrastructure (VI) or application to shrink and grow upon request. This configuration is done without human intervention or support to write configuration files or restart services. A VI embraces the allocated compute, storage, and network resources. For example new instances of the application are integrated in the overall system and closing instances lead not to data lost. Increase storage capacity will be utilized and removed computing capacity (CPU) will not crash the application. For example, a FNS is automatically setup between a new and running VMs,

- Secondly, the adaptation is a reactive system. Decision logic provides a new deployment configuration upon monitoring data. For a distributed application, online performance data provides the input to decide where to place application's VMs.

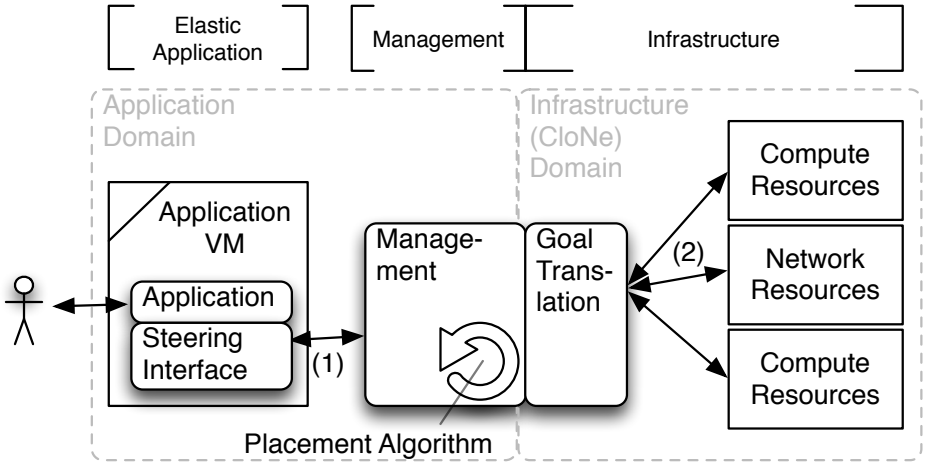


Figure 5.5: Architecture extensions and essential components.

5.4.2 A Possible Implementation Approach for Dynamic Adaptation

These two capabilities are implemented in 4 parts, as shown in Figure 5.5. The Figure depicts parts of the overall architecture described in this deliverable, put in action for dynamic load adaptation.

Elastic Application An elastic application is capable of reacting adequately on changes of the VI. Additionally, the application has to provide online performance data to the *Management* component.

Management This component decides upon performance, monitoring data, and adjusts the VI accordingly. The adjustment is based upon a *Placement Algorithm*. The adjustments are applied through *Infrastructure* providers. The running application is updated about the changes.

Infrastructure The different infrastructures create, manage, and connect the virtual compute and network resources of a VI. CloNe is capable of supporting anything needed.

Placement Algorithm The algorithm collects application's online performance data and infrastructure monitoring data as the input. From this input the algorithm computes the new layout of the VI: "At which location how many of which VMs are running."

In the remaining part of this section we will describe these 4 components in more detail and how they relate to other parts of the SAIL project spanning beyond the scope of CloNe. We offer a more detailed explanation on how these components work and what their implementation-level dependencies are. Finally, we show that such design is actually feasible in our prototype, adding weight to the validity of our architecture and its realisation potential.

Elastic Application

An elastic application is also a distributed application, which consist of distributed components or instances (in VMs) realizing the service or functionality of that application. An elastic application is seen as untouched by end users, while the amount of available resources changes. This appears on two different levels: either new resources like VMs are added/remove or old resources like storage are increased/decreased. Obviously, simple stateless web-server are elastic. Such applications can already be scaled up and down with Amazon's CloudWatch[79].

Using the same technology for stateful applications will cause malfunction. They lose their state information, business or user data, upon scaling down and shutdown of instances. Traditional applications cannot handle dynamic adjustment of resources, like storage, memory, or amount of CPUs. Only the application manages the available resources and has to decide how to use them efficiently. Thus the application needs to be informed about changes of available resources and amount of resources. These application level adjustments have more potential as guest operating system or hypervisor mechanisms.

For simple applications, newly launched instances do not need context information like "which other instances are running". Other applications are similar to the three tier application described at the beginning of this section which includes interactions between different components. We need a way to share or retrieve context information. In special cases like a virtual private network this can be done by broadcast. In other situations, DNS resolution with customized DNS entries will help. This is an announced Amazon service, Route 53 [80]. However, these solutions will only work in certain situations. A more flexible way is to configure the VM after it is booted and eventually before the application is started. This way context information can be embedded in the application configuration or a running application can be reconfigured.

As a generalization, a VM/application can be informed or triggered about any event related to the changes of their VI: upon start, resume, migration to (re)configure the application, upon stop or suspend to rescue the state. Technically, the VM and the application are informed via a *steering interface*. Either the application can implement this interface or a steering daemon is installed as separate software. The daemon has to launch upon start and mediates between the application and the external component using the steering interface.

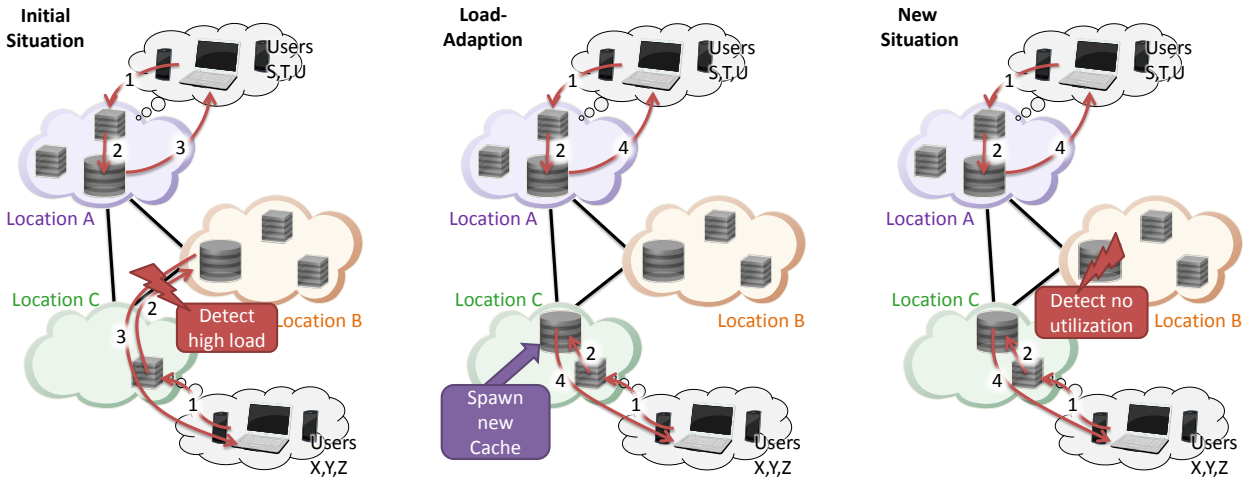
In our case, the Management component informs the application through the steering interface about changes of their VI, which is shown in Figure 5.5 as arrow (1). The Management component which computes the VI has all necessary information about the VI available.

Additionally, the application needs to provide online performance data, so that the management component can decide upon an appropriate placement. Either the application or the steering daemon, which access application's data or logs, has to provide such data. In our case, this results in a two-way communication between the VM/application and the Management component.

Management

The Management component gathers performance data from the application, monitoring data from the infrastructure and discovers potential cloud location. Upon this data, the Placement Algorithm decides, at which cloud location how many resources need to be allocated for the application. The actual decision logic and optimization goal is encapsulated in the Placement Algorithm, so that the Management component serves as a generic frame.

The Management component can be seen as a high-level inter-provider goal translator in the context of the SAIL architecture. The higher level is the result of an abstract description of the application as a kind of template and an abstract optimization of the placement of individual VMs (realized by the Placement Algorithm). In contrast a lower level goal translation contains every VM/application instance and every connection (FNS) to be allocated, which for example can be specified in VXDL (c.f. Section 5.3.1 Goal Translation). The Management component implicitly does a low-level goal translation and interacts with the Virtual Infrastructure Provider directly, or a low-level goal translator like CloudWeaver is intermediary utilized. Because of this blur, both functionalities, the high-level and low-level goal translation, are drawn side-by-side in Figure 5.5.



(a) Detection of high load (b) React by spawning a new cache (c) Detect low utilization

Figure 5.6: Scenario C: Load-adaptive NetInf deployment

Infrastructure

The existing part of the CloNe architecture implements the necessary features to realize such an inter-provider deployment. The Management component interacts with a separate broker, goal translator, or directly with the virtual infrastructures via the north bound CloNe interfaces. CloNe does the necessary inter-provider negotiation internally, so the Management component is left out.

Placement Algorithm

The Placement Algorithm is very specific to the optimization goal, the application type and the data provided. Currently, two concrete incarnations are analysed more closely:

Response Time For interactive (computing intensive) applications, like the video cut application, the responsiveness of user interactions is important. An interaction lag renders the application nearly useless. For this an appropriate algorithm is in development, where cloud locations close by users are favoured.

Inter-provider Link Both CDN or NetInf provider minimizing their costs. Transfer costs (for example across interprovider links) are saved by having a cache on the other site, so data has to cross only once. Together with WPB, we envision a dynamic allocation of such caches. This way, the system can adapt to the location from where the downloads are started. As a result, no upfront wide area cache deployment is necessary.

While the architecture is in place and much of the prototype development is done, the research for the Placement Algorithms are still in progress.

5.4.2.1 Proof of the Feasibility of the Proposed Load Adaptation Features

The system design was prototyped to test an adaptive NetInf deployment across data centres. While the different scenarios are described in detail in Chapter 4 of D.A.9[62], the last scenario c describes the load-adaptive placement of NEC NetInf Router Platform (NNRP) nodes.

We envision a situation where requests for content are already served by the nearest content cache (Location A and C). Customers from Location C are now requesting content, which is not

available at Location C but can be served from Location B. As a result, a lot of (inter-provider) traffic is caused (Figure 5.6(a)). CloNe can detect the high load on the inter-location link and react with a pre-defined, NetInf-specific action. This action can be part of the deployment request from the NetInf provider. In our example this action is to add a new cache in Location C (Figure 5.6(b)). As a result the requested objects will also be cached in Location C. After some time no further inter-provider traffic is produced (except for updates). Because CloNe not only monitors the load on the inter-provider links but also the CPU load of the VMs, it now detects that the cache in Location B is no longer required (Figure 5.6(c)). Again using a predefined policy, this un-utilized node at Location B is shut down to save cost.

6 Application use-cases played out

The infrastructure services provided by CloNe are meant to be used by a tenant. In general terms, the tenant is a person or entity that wishes to deploy virtual infrastructure with specific requirements. On top of that virtual infrastructure, an application or a set of connected applications will be deployed. The application is provided, deployed and maintained by the tenant. Nevertheless, the Infrastructure Service Provider must abide to infrastructure requirements provided by the tenant, which are obviously application specific requirements.

The objective of this chapter is to show how the tenant interacts with the CloNe service, and how the internal components of the architecture are used to implement the service demanded by the tenant. The two scenarios of CloNe, namely Dynamic Enterprise and Elastic Video Distribution will be used to show case two ways in which the interaction with the tenant may happen. Although the initial placement of the nodes follows the same steps in both cases, adaptation to the application elasticity in the second use case needs further interactions and information exchange between infrastructure provider and decomposer. While the Dynamic Enterprise use case focuses on the integrated provisioning of on-demand networking services connecting IaaS, the Elastic Video Distribution focuses on the automated scaling of an application on the infrastructure to meet a specific objective. We described the main actors in CloNe while describing the overall architecture in the second chapter. While describing the use cases below, these roles are played out.

6.1 Dynamic enterprise

The dynamic enterprise scenario is about on-demand provisioning of computing and networking resources across data centres and networks. Enterprises need not only an easy way to add and remove resource (both computing and networking); they may need to extend their business systems to new remote locations (either temporarily or permanently). Moreover, the enterprise tenant expects a higher level of reliability, performance, security and isolation. The global presence of large enterprises implies that CloNe should be able to work across different providers.

CloNe enables the dynamic enterprise scenario by providing IaaS from data centres integrated with an FNS service provided by a network operator. Current existing solutions focus on the provisioning of resources within a data centre. CloNe is able to integrate wide area networking resources into the equation, providing an end-to-end solution.

6.1.1 Tenant interaction

The steps the tenant should go through to successfully deploy its application on the CloNe infrastructure are:

- Identification of application to be deployed on CloNe
- Investigating how the application connects to rest of system
- Specification of a virtual infrastructure including target application
- Sending request for virtual infrastructure to Infrastructure Service Provider

- Receiving response to request, acknowledging infrastructure deployed
- Deploying applications and connecting existing systems
- Requesting different amount of resources

The first step of the process consists on the identification of the application to be deployed on the infrastructure. Applications with varying demand are very suited since that will yield lower CAPEX costs. There are likely business reasons that will influence this process, such as liability, security, amongst others. It is also possible to use the CloNe solution to provide WAN services only. That is the case when CloNe is used to connect two existing private data centres located in the enterprises premises.

Before deploying its infrastructure on CloNe, the enterprise tenant must have a good understanding of how the target application interacts with existing systems. Typical enterprise applications are tightly connected to other internal systems (e.g., e-commerce connected to billing, order handling, stock management, delivery system, and suppliers). Rarely an enterprise application runs stand-alone without any dependencies. The tenant must know how those systems are connected (topology), the type of connectivity utilized (addressing), the average amount of data exchanged (capacity), and other application requirements (e.g., maximum delay between servers). It is assumed that to minimize application impact, the application will have the same type of infrastructure as it did when running it internally.

Based on information gathered in the previous step, the tenant is then able to specify the desired virtual infrastructure. The virtual infrastructure is specified using a descriptive language such as VXDL or a graphical user interface intended to generate such a file. VXDL allows one to specify full virtual infrastructures composed of virtual nodes and links. Temporal changes in the infrastructure can be specified, with demand changing through time. Specific legal requirements are included, e.g., need to execute a service under a given jurisdiction. This file is an SLA between the tenant and the Infrastructure Service Provider.

That file is then sent to an Infrastructure Service Provider, which verifies if the request can be satisfied. It may be necessary to use the concept of Delegation to provide the full infrastructure to the tenant. Regardless, that will be transparent to the tenant. When the process (Section 6.1.2) is finished, the tenant receives a response, if positive it includes the Virtual Infrastructure Identifier that is used in further requests. When the response is received, the infrastructure is up and running and the tenant should be able to use it.

The next step is the deployment of the actual application. If the virtual infrastructure is composed of standard types of virtual machines provided by the ISP (e.g., a Linux VM, kernel X.Y.Z), the tenant can probe CloNe to obtain IP addresses and credentials to access these servers. With access to that virtual machine the tenant should be able to copy and install any type of software needed. However, if the tenant had specified the use of private owned VM images, those should be first uploaded. Once uploaded, it should be possible to notify the CloNe infrastructure of its availability.

The tenant should make sure that any private owned part of the infrastructure abides to the specification sent to the ISP. That is, existing parts of the service that will interact with the cloud should be accessible and connected to the attachment points specified in the file.

At this point, the tenant application should be running and connected to the rest of the system. If changes are to be performed, the Infrastructure ID should be used. The customer shall be able to change any part of the infrastructure, including, changing capacity of links, changing topology of the infrastructure, and changing type of connectivity utilized, amongst others. This is done by submitting an updated VXDL file that specifies the new needs of the tenant.

6.1.2 Components involved

The concepts and components architected in CloNe work together to implement the distributed virtual infrastructure across different providers. Once the request from the tenant is received, the following steps are taken:

- The VXDL request is parsed and the Decomposer is the entity in the architecture that is responsible for determining where the infrastructure is to be deployed as the placement logic resides in this component. Most of the envisioned virtual infrastructures will involve more than one provider and thus it is the Decomposer's responsibility to break down the request onto smaller pieces of infrastructure that will be deployed by different Infrastructure Service Providers. All of the tenant's requirements must be met or the request has to be denied.
- Delegation: the process of passing on requests to other providers for deployment. This is done by calling the Infrastructure Service Interface of the respective providers. These are interfaces that allow for the creation of virtual resources (e.g., RESTful interfaces such as OCCI/OCNI). An Virtual Infrastructure Identifier that represents the infrastructure is generated and passed along with the requests.
- In case one of the providers is not able to fulfil the request, a new provider needs to be found that meets the tenants requirements. If none is found, the request is denied.
- Each one of the providers may decide to break down the request into smaller pieces of infrastructure and use Delegation again. When the provider decides to deploy the virtual infrastructure locally, it will use the Resource Allocation function to allocate virtual machines, storage and network. All of the tenants requirements must be fulfilled. The provider implementing the infrastructure signals to the requesting provider the fact that the infrastructure is ready to be used.
- Once all of the pieces of infrastructure are deployed it is time to connect them together. In order for that to happen, a provider needs to know whom it shall connect its piece of infrastructure to. Through the Infrastructure Service Interface, using the pre-defined Virtual Infrastructure ID, the peer providers are made aware of each other. The process is of course recursively applied if further Delegation has been used.
- The involved providers use the Distributed Control Plane to connect their pieces of infrastructure. DCP is used for link negotiation between the two involved providers. For example, the characteristics of the link are agreed (hose model), encapsulation schemes (e.g., VLAN, GRE), routes that should be announced (OSPF routes), amongst others.
- Once DCP has finished agreeing upon connections, the full Virtual Infrastructure is ready to be used. The Virtual Infrastructure ID is returned to the tenant.

6.2 Elastic Video Distribution

The Elastic Video Distribution scenario is not different from any other distributed application. Therefore, this use case focuses on the deployment of distributed application at the edge of the network, leveraging on distributed computational resources made available by a network operator. Computing resources are geographically distributed in the operator network in a more fine-grained fashion than today's data centre cloud deployments. The motivation for doing so is to leverage on the closeness to end-user offering enhanced QoE. Moreover, such distributed deployment will

reduce the stress on networks, since content can be pushed to the edge servers when the core network is less congested (so called *pre-caching*).

This scenario is made more complex since CloNe aims at automatic scalability of the application based on end-users demand. The CloNe solution is able to start new instances of Content Servers in an automated fashion once demand increases.

6.2.1 Tenant interaction

Differently from the scenario described in Section 6.1 where the application is unaware of the infrastructure, in this case the application needs to interact with the infrastructure much more closely. This is because the automated deployment of servers needs to be notified to the application that needs to take action to include new servers in the pool of resources. For example, in the case of video delivery, if the CloNe platform creates a new Video Streamer, the application should add that server to its DNS so clients can start using it.

Besides the IaaS offering seen in the previous scenario (Section 6.1) providing a framework for managing and interconnecting virtual resources spanning multiple providers; CloNe also allows end-users to send richer requests specifying how the to-be-deployed application will react upon load variations at any of its encompassing components.

The steps through which the tenant should go through to successfully deploy an application capable of scaling up or down automatically on the CloNe infrastructure are:

- Identification of the elastic application to be deployed on CloNe (e.g. video distribution)
- Identification of the elastic components encompassing (components that will be scaled up or down based on end-users demand) the application.
- Specification of a virtual infrastructure including: 1. the virtual images to be used; 2. the resource requirements; and 3. the policies on how to react on changes on the to be monitored KPIs (load data).
- Sending request for virtual infrastructure to Infrastructure Service Provider
- Receiving response to request, acknowledging infrastructure deployed

6.2.2 Components involved

A CloNe provider offers a single point of access (Figure 6.1) to deploy elastic applications. As it is seen in this schematic picture, the monitoring and topology information are received from different network and data centre providers from one side and the application request is received via VXDL file from other side. This CloNe provider can divide service deployment requests into small chunks and delegate these to other more appropriate CloNe providers. Before any request from the tenant is received, this management component discovers and knows the topology of all CloNe infrastructure providers. Once the request from the tenant is received, the same steps described in Section 6.1 are taken in order to perform the initial placement of the elastic application.

Once the application has been initially deployed, and upon regularly collected and received KPIs coming from both the infrastructure and the application layers, a placement logic decides automatically to deploy new or shut down old VMs (e.g. intermediate or caching nodes in a video distribution overlay). This decision is made based on the policies on how to react on changes on the monitored KPIs specified in the initial VXDL request.

The process of deploying new or shutting down old VMs is done by calling the Infrastructure Service Interface of the target infrastructure providers. These are interfaces that allow for the creation of virtual resources (e.g., RESTful interfaces such as OCCI/OCNI).

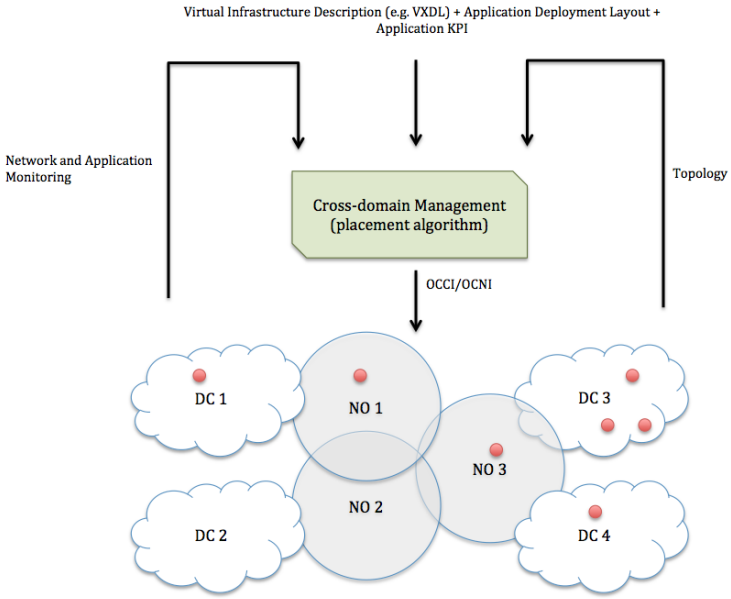


Figure 6.1: Elastic Application Deployment

The components involved in the use cases described above are the orchestration and distributed control components that are required to realize these two use cases. However, as it was noticed in previous chapters, there are several optimization, management, security and enhancement components such as optimized resource management or network resource control that are not mentioned here. These components are invoked by infrastructure service providers or data centre and network management specifically when applicable.

7 Conclusions

The architecture presented in this document is the result of an iterative design and prototyping approach. The CloNe work package of the SAIL project developed the initial Cloud Networking Architecture during its first year and used it as the basis of multiple prototyping and simulation efforts. Some of these prototypes implemented specific management and security functions and some developed complete systems involving multiple data centres and networks spread across Europe. These prototypes are described in deliverable *D-5.2 Description of Final Prototype* [3] and the various published literature referenced throughout this document.

Experience gained through prototyping lead to refinements in the concepts, more detail in the interactions, and a better understanding of the layering of the architecture itself. The Flash Network Slice (FNS) concept has been expanded to include more detail in how constraints and network functions can be included in its definition. This also provided a better understanding of the division between technology independent and implementation specific aspects. The building blocks and mappings described in Chapter 3 are the result of practical implementation using multiple network technologies.

The functions and interactions of the infrastructure service provider have been clearly separated out into the service, inter-provider, and intra-provider layers described in Chapter 2. The separation of concerns between the infrastructure service itself and the interactions of the Distributed Control Plane (DCP) has been properly established with the division between the service layer and the inter-provider layer. In this division the service layer is dealing with abstract infrastructure definitions and delegation between providers based on service level objectives, and the cross provider layer is dealing with negotiation of configuration details, discovery of capabilities, and implementation of security controls. This distinction became clear during the implementation of the complete prototype described in [3].

The functions that comprise the management aspect of the architecture were re-factored to identify resource and performance monitoring and resource control as independent functions. These two were previously considered integral to fault management and resource management, but are now viewed as independent functions that support not only resource management and fault management but also other functions. Additionally the Distributed Knowledge Plane (DKP) has been removed as a concept. The DKP provided distributed information access. This is now viewed as an artefact of a distributed implementation design choice and not an architectural concept.

The security aspect has been expanded to accommodate the service layer delegation concept. In particular it is recognised that the authorisation function will need to operate in the presence of delegation chains due to user access right delegation and provider implementation delegation.

7.1 Contributions

The main focus of the CloNe work package has been to gain an understanding of cloud networking: the combined provision of network, compute, and storage resources across diverse infrastructure service providers. The basis of this understanding is the concept of the FNS network resource and how it relates to the cloud computing Infrastructure as a Service (IaaS) paradigm.

Additionally we have explored delegation as a means to organise management of virtual infrastructure in multiple administrative domains. This approach is suitable for use within a single

organisation, such as a network operator or a data centre operator, as a means to organise internal management systems, or across organisational boundaries as a means to support virtual infrastructure management across providers. Delegation will allow for an ecosystem of cloud providers to utilize each other's resources while satisfying the requirements of the cloud customer.

The CloNe architecture proposed in this document goes beyond the state of the art in the area of IaaS. The FNS provides an abstraction of a network service that fits the model of cloud computing and that addresses one of the missing parts of existing cloud services: "the ability to define and manage a network service that spans providers". The FNS provides dynamic connectivity services with a defined level of performance and reliability expected by enterprise applications to be deployed in the cloud. The architecture allows users to specify measurable performance goals associated to resources allocated in the infrastructure.

Yet another new concept is the deployment of computing and storage resources within the network. These resources allow a finer level of distribution of service than the one provided by existing data centres. Decisions to place and scale compute and storage resources can be combined by the providers, with their understanding of network performance and capacity, to provide better service to their users and better optimisation of their infrastructure.

The concepts developed for the CloNe architecture have contributed to the development of open source with libNetVirt [81] and pyOCNI [13], and to standardisation with the VXDL [82] and OCNI [13] infrastructure description languages and interfaces.

7.2 Closing Remarks

The CloNe architecture has been used to build a successful prototype of a cloud system interconnecting services running across data centres and operator networks and therefore can be the base for materializing such service by actual data center and network providers today. Businesses small and large are using cloud infrastructure to host their IT services and to consume externally provided IT services. They need to integrate these with their own infrastructure to create secure, managed environments. At present this integration activity is manual and providers understand that it is not sufficient. Experience has shown that the FNS and DCP developed by CloNe plays an important role in enabling these integrated infrastructures.

Additionally, the ability to deploy services across and within networks, as a way to take advantage of network proximity to customers and their (potentially mobile) end users, is developing as a means to support on-demand services from bulk-data transfer, to streaming media and on-line gaming.

It is clear that the concepts and mechanism developed in the CloNe architecture enable real business needs of cloud computing to be met in the on-going evolution of the Internet.

List of Figures

List of Tables

List of Acronyms

AMQP Advanced Message Queuing Protocol

API Application Programming Interface

CDN Content Distribution Network

CDMI Cloud Data Management Interface

CloNe Cloud Networking

CMBS Cloud Message Brokering Service

CPU Central Processing Unit

DCP Distributed Control Plane

DKP Distributed Knowledge Plane

EDA Event Driven Architecture

GRE Generic Routing Encapsulation

FNS Flash Network Slice

IaaS Infrastructure as a Service

IDS Intrusion Detection System

IPsec Internet Protocol Security

JMS Java Message Service

KDD Knowledge Discovery and Data Mining

LNP Link Negotiation Protocol

MPLS Multiprotocol Label Switching

NetInf Network of Information

NNRP NEC NetInf Router Platform

NIC Network Interface Card

OCCI Open Cloud Computing Interface

OCNI Open Cloud Networking Interface

QoE Quality of Experience

QoS Quality of Service

S3	Simple Storage Service
SaaS	Software as a Service
SAIL	Scalable Adaptive Internet Solutions
SAML	Security Assertion Markup Language
SIEM	Security Information and Event Management
SLA	Service Level Agreement
SPML	Service Provisioning Markup Language
SSL	Secure Socket Layer
SVM	Support Vector Machine
TPM	Trusted Platform Module
UMA	User Managed Access
URI	Universal Resource Identifier
VI	Virtual Infrastructure
VLAN	Virtual Local Area Network
VM	Virtual Machine
VN	Virtual Network
VPLS	Virtual Private LAN Service
VPN	Virtual Private Network
VXDL	Virtual private eXecution infrastructure Description Language
WAN	Wide Area Network
WPD	Work Package D
XML	Extensible Markup Language

References

[1] The SAIL project web site. `http://www.sail-project.eu/`.

[2] SAIL Consortium. D-5.2 (d-d.1) cloud network architecture description. Technical report, ICT-SAIL project 257448, July 2011.

[3] The SAIL Consortium. D-D.2: Description of the implemented prototype. Technical report, FP7-ICT-2009-5-257448-SAIL, 2012.

[4] Edwall, Thomas. D-2.1 (D-A.1) Description of project wide scenarios and use cases. Technical report, FP7-ICT-2009-5-257448-SAIL, 2011.

[5] ACM SIGCOMM 2012 Conference, SIGCOMM '12, Helsinki, Finland, August 2012.

[6] GEYSERS: Generalised Architecture for Dynamic Infrastructure Services. http://www.geysers.eu/.

[7] Future Network Mobile Summit 2012, Berlin, Germany. `http://www.futurenetworksummit.eu/2012/`.

[8] Moritz Steiner, Bob Gaglianello Gaglianello, Vijay K. Gurbani, Volker Hilt, William D. Roome, Michael Scharf, and Thomas Voith. Network-aware service placement in a distributed cloud environment. In *Proceedings of the ACM SIGCOMM 2012 Conference*, pages 73–74, 2012.

[9] Peter Xiang Gao, Andrew R. Curtis, Bernard Wong, and Srinivasan Keshav. Its not easy being green. In *Proceedings of the ACM SIGCOMM 2012 Conference*, SIGCOMM 12, page 211222, New York, NY, USA, 2012. ACM.

[10] Hans Lindgren, Fetahi Wuhib, and Rolf Stadler. Dynamic resource allocation with management objectives : Implementation for an openstack cloud. In *Proceedings of 8th International Conference on Network and Service Management (CNSM 2012)*, 2012.

[11] Quantum L3 Abstractions and API Framework , December 2011. https://blueprints.launchpad.net/quantum/+spec/quantum-l3-api.

[12] Guilherme Koslovski, Pascale Vicat-Blanc Primet, and Andrea Schwertner Charão. VXDL: Virtual Resources and Interconnection Networks Description Language. In *The Second International Conference on Networks for Grid Applications (GridNets 2008)*, Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering. Springer Berlin Heidelberg, Oct. 2008.

[13] pyOCNI - a Python implementation of an extended OCCI with a JSON serialization and a cloud networking extension. Online URL: http://occi-wg.org/2012/02/20/occi-pyocni/.

[14] Ralf Nyrén, Andy Edmonds, Alexander Papaspyrou, and Thijs Metsch. Open Cloud Computing Interface – Core. GFD-P-R.183, April 2011.

[15] Thijs Metsch and Andy Edmonds. Open Cloud Computing Interface – Infrastructure. GFD-P-R.184, April 2011.

[16] OpenNebula. The open source solution for data center virtualization. http://www.opennebula.org/.

[17] Rackspace Cloud Computing. Openstack cloud software. http://www.openstack.org/.

[18] SAIL Consortium. D-3.1 (d-b.1) the network of information: Architecture and applications. Technical report, ICT-SAIL project 257448, July 2011.

[19] Libvirt. `http://libvirt.org/`.

[20] A.G. Prieto, D. Gillblad, R. Steinert, and A. Miron. Toward decentralized probabilistic management. *Communications Magazine, IEEE*, 49(7):80 –86, July 2011.

[21] Schoo, Peter and Fusenig, Volker and Souza, Victor and Melo, Márcio and Murray, Paul and Debar, Herve and Medhioub,Houssem and Zeghlache,Djamal. Challenges for Cloud Networking Security. In Kostas Pentikousis, Ramón Agüero Calvo, Marta García-Arranz, and Symeon Papavassiliou, editors, *MONAMI*, volume 68 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, pages 298–313. Springer, 2010.

[22] Eileen Forrester. Aligning Security Management Processes with CMMI. *Defense*, pages 1–9, 2005.

[23] Giles Hogben. ENISA Cloud Computing Security Strategy. *Terena*, 2011.

[24] Clinch, Jim. ITIL V3 and Information Security by Jim Clinch. *Information Security*, pages 1–40, 2009.

[25] Volker Fusenig and Ayush Sharma. Security architecture for cloud networking. In *Computing, Networking and Communications (ICNC), 2012 International Conference on*, pages 45–49, 30 2012-feb. 2 2012.

[26] Horecker, B L and Stadtman, E REditors, editor. *OASIS Service Provisioning Markup Language (SPML) Version 2*. Academic Press, 2006.

[27] Eric Dubuis. SAML Assertions. *Security*, 2011.

[28] Dan Wang and Dengguo Feng. A Hypervisor-Based Secure Storage Scheme. In *Networks Security Wireless Communications and Trusted Computing (NSWCTC), 2010 Second International Conference on*, volume 1, pages 81 –86, april 2010.

[29] CloudAudit, June 2012. http://cloudaudit.org/.

[30] E. Rosen and Y. Rekhter. BGP/MPLS IP Virtual Private Networks (VPNs), RFC 4364.

[31] W. Augustyn and Y. Serbest. Framework for Layer 2 Virtual Private Networks (L2VPNs), RFC 4664.

[32] W. Augustyn and Y. Serbest. Service Requirements for Layer 2 Provider-Provisioned Virtual Private Networks, RFC 4665.

[33] R. Santitoro. Metro Ethernet Services - A Technical Overview.

[34] K. Kompella and Y. Rekhter. Virtual Private LAN Service (VPLS) Using BGP for Auto-Discovery and Signaling, RFC 4761.

[35] M. Lasserre and V. Kompella. Virtual Private LAN Service (VPLS) Using Label Distribution Protocol (LDP) Signaling, RFC 4762.

[36] Guilherme Koslovski, Pascale Vicat-Blanc Primet, and Andrea Schwertner Charão. VXDL: Virtual Resources and Interconnection Networks Description Language. In *The Second International Conference on Networks for Grid Applications (GridNets 2008)*, Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, Beijing, China, Oct. 2008. Springer Berlin Heidelberg.

[37] Right Scale: Set up Autoscaling using Voting Tags. Available: `http://support.rightscale.com/`.

[38] Daniel Morn, Luis M. Vaquero, and Fermin Galn. Elastically ruling the cloud: Specifying application's behavior in federated clouds. *Cloud Computing, IEEE International Conference on*, 0:89–96, 2011.

[39] Luis Rodero-Merino, Luis M. Vaquero, Victor Gil, Fermín Galán, Javier Fontán, Rubén S. Montero, and Ignacio M. Llorente. From infrastructure delivery to service management in clouds. *Future Gener. Comput. Syst.*, 26:1226–1240, October 2010.

[40] Luis M. Vaquero, Daniel Moran, Fermin Galan, and Jose M. Alcaraz-Calero. Towards Runtime Reconfiguration of Application Control Policies in the Cloud. *Journal of Network and Systems Management*, 1(1), 2011.

[41] Amazon Auto Scaling, June 2012. Available: `http://aws.amazon.com/autoscaling/`.

[42] Scalr. Available: `http://www.scalr.net/`.

[43] B. Bjurling and R. Steinert and D. Gillblad. Translation of Probabilistic QoS in Hierarchic and Decentralized Settings. In *The 13th Asia-Pacific Network Operations and Management Symposium*, Taipei, Taiwan, 2011.

[44] R. Steinert, S. Gestrelius, and D. Gillblad. A distributed spatio-temporal event correlation protocol for multi-layer virtual networks. In *Global Telecommunications Conference (GLOBECOM 2011), 2011 IEEE*, pages 1–5. IEEE, 2011.

[45] R. Steinert and D. Gillblad. Link delay modeling and direct localization of performance degradations in transport networks. In *Submitted to INFOCOM 2013*. IEEE, 2012.

[46] Y. Zhu and M. Ammar. Algorithms for assigning substrate network resources to virtual network components. In *INFOCOM 2006. 25th IEEE International Conference on Computer Communications. Proceedings*, pages 1–12, 2006.

[47] M. Yu, Y. Yi, J. Rexford, and M. Chiang. Rethinking virtual network embedding: Substrate support for path splitting and migration. *ACM SIGCOMM Computer Communication Review*, 38(2):17–29, 2008.

[48] N.M.M.K. Chowdhury, M.R. Rahman, and R. Boutaba. Virtual network embedding with coordinated node and link mapping. In *INFOCOM 2009, IEEE*, pages 783–791, 2009.

[49] J. Nogueira, M. Melo, J. Carapinha, and S. Sargento. Virtual network mapping into heterogeneous substrate networks. In *IEEE Symposium on Computers and Communications (ISCC) 2011*, June 2011.

[50] Michael Isard. Autopilot: automatic data center management. *SIGOPS Oper. Syst. Rev.*, 41(2):60–67, April 2007.

[51] Márcio Melo, Jorge Carapinha, Susana Sargento, Luis Torres, Phuong Nga Tran, Ulrich Killat, and Andreas Timm-Giel. Virtual network mapping - an optimization problem. In Kostas Pentikousis, Rui Aguiar, Susana Sargento, Ramón Agüero, Ozgur Akan, Paolo Bellavista, Jiannong Cao, Falko Dressler, Domenico Ferrari, Mario Gerla, Hisashi Kobayashi, Sergio Palazzo, Sartaj Sahni, Xuemin (Sherman) Shen, Mircea Stan, Jia Xiaohua, Albert Zomaya, and Geoffrey Coulson, editors, *Mobile Networks and Management*, volume 97 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, pages 187–200. Springer Berlin Heidelberg, 2012.

[52] L.A. Wolsey. Integer programming. *IIE Transactions*, 32:273–285, 2000.

[53] S. Even, A. Itai, and A. Shamir. On the complexity of time table and multi-commodity flow problems. In *16th Annual Symposium on Foundations of Computer Science*, page 184–193, 1975.

[54] M. Pióro and D. Medhi. *Routing, Flow, and Capacity Design in Communication and Computer Networks*. Elsevier/Morgan Kaufmann, 2004.

[55] Fetahi Wuhib, Rolf Stadler, and Mike Spreitzer. Gossip-based resource management for cloud environments. In *International Conference on Network and Service Management*, October 2010.

[56] Rerngvit Yanggratoke, Fetahi Wuhib, and Rolf Stadler. Gossip-based resource allocation for green computing in large clouds. In *International Conference on Network and Service Management*, October 2011.

[57] F. Wuhib, R. Stadler, and M. Spreitzer. A gossip protocol for dynamic resource management in large cloud environments. *Network and Service Management, IEEE Transactions on*, 9(2):213 –225, june 2012.

[58] J. Soares, R. Monteiro, M. Carapinha, J. Melo, and S. Sargento. Resource allocation in the network operator's cloud: A virtualization approach. In *Computers and Communications (ISCC), 2012 IEEE Symposium on*, pages 000800 –000805, july 2012.

[59] Amir Nahir, Ariel Orda, and Danny Raz. Distributed oblivious load balancing using prioritized job replication. Technical report, Technion, Israel Institute of Technology, April 2011.

[60] Paulo Gonçalves, Shubabrata Roy, Thomas Begin, and Patrick Loiseau. Dynamic resource management in clouds: A probabilistic approach. *IEICE Transactions on Communications, special session on Networking Technologies for Cloud Services*, 2012.

[61] Eucalyptus. The open source cloud platform. http://open.eucalyptus.com/.

[62] The SAIL Consortium. D-2.9: (D.A.9) Description of the Overall Prototyping Use cases, Scenarios and Integration Points. Technical report, FP7-ICT-2009-5-257448-SAIL, 2012.

[63] Juliano Araujo Wickboldt, Lisandro Zambenedetti Granville, Dominique Dudkowski, and Marcus Brunner. Hyfs manager: A hybrid flash slice manager. In *Presented in Student Demo Contest of the 13th IEEE/IFIP Network Operations and Management Symposium (NOMS 2012)*, April 2012.

[64] Juliano Araujo Wickboldt, Lisandro Zambenedetti Granville, Fabian Schneider, Dominique Dudkowski, and Marcus Brunner. Network-centric design of customizable resource allocation for distributed cloud infrastructures. In *Accepted as short paper at the 8th International Conference on Network and Service Management (CNSM 2012)*, October 2012.

[65] A. Kannan, A. Sharma, Jr. G.Q. Maguire, and P. Schoo. Genetic Algoritm based Feature Selection Algorithm for Effective Intrusion Detection in Cloud Networks. In *Submitted for ICDM 2012 (IEEE internation conference on Data Mining)*, 2012.

[66] I. Aguirre and S. Alonso. Improving the automation of security information management: A collaborative approach. *Security Privacy, IEEE*, 10(1):55 –59, jan.-feb. 2012.

[67] Yuteng Guo, Beizhan Wang, Xinxing Zhao, Xiaobiao Xie, Lida Lin, and Qingda Zhou. Feature selection based on Rough set and modified genetic algorithm for intrusion detection. In *Computer Science and Education (ICCSE), 2010 5th International Conference on*, pages 1441 –1446, aug. 2010.

[68] Gary Stein, Bing Chen, Annie S. Wu, and Kien A. Hua. Decision tree classifier for network intrusion detection with GA-based feature selection. In *Proceedings of the 43rd annual Southeast regional conference - Volume 2*, ACM-SE 43, pages 136–141, New York, NY, USA, 2005. ACM.

[69] Cao Li-ying, Zhang Xiao-xian, Liu He, and Chen Gui-fen. A network intrusion detection method based on combined model. In *Mechatronic Science, Electric Engineering and Computer (MEC), 2011 International Conference on*, pages 254 –257, aug. 2011.

[70] M. Tavallaee, E. Bagheri, Wei Lu, and A.A. Ghorbani. A detailed analysis of the KDD CUP 99 data set. In *Computational Intelligence for Security and Defense Applications, 2009. CISDA 2009. IEEE Symposium on*, pages 1 –6, july 2009.

[71] I.A. Tndel, J. Jensen, and L. Rstad. Combining Misuse Cases with Attack Trees and Security Activity Models. In *Availability, Reliability, and Security, 2010. ARES '10 International Conference on*, pages 438 –445, feb. 2010.

[72] J.A. Ingalsbe, L. Kunimatsu, T. Baeten, and N.R. Mead. Threat Modeling: Diving into the Deep End. *Software, IEEE*, 25(1):28 –34, jan.-feb. 2008.

[73] R. Ahmad and L. Janczewski. Governance life cycle framework for managing security in public cloud: From user perspective. In *Cloud Computing (CLOUD), 2011 IEEE International Conference on*, pages 372 –379, july 2011.

[74] M. Nottingham and R. Sayre. The Atom Syndication Format, RFC 4287.

[75] Maciej P. Machulak, Lukasz Moreń, and Aad van Moorsel. Design and implementation of user-managed access framework for web 2.0 applications. In *Proceedings of the 5th International Workshop on Middleware for Service Oriented Computing*, MW4SOC '10, pages 1–6, New York, NY, USA, 2010. ACM.

[76] Suhas Pai, Yash Sharma, Sunil Kumar, Radhika M. Pai, and Sanjay Singh. Formal Verification of OAuth 2.0 Using Alloy Framework. In *Proceedings of the 2011 International Conference on Communication Systems and Network Technologies*, CSNT '11, pages 655–659, Washington, DC, USA, 2011. IEEE Computer Society.

[77] Jose M. Alcaraz Calero, Nigel Edwards, Johannes Kirschnick, Lawrence Wilcock, and Mike Wray. Toward a Multi-Tenancy Authorization System for Cloud Services. *IEEE Security and Privacy*, 8(6):48–55, November 2010.

[78] Rodrigo N. Calheiros, Christian Vecchiola, Dileban Karunamoorthy, and Rajkumar Buyya. The aneka platform and qos-driven resource provisioning for elastic applications on hybrid clouds. *Future Gener. Comput. Syst.*, 28(6):861–870, June 2012.

[79] Amazon CloudWatch, June 2012. Available: `http://aws.amazon.com/cloudwatch/`.

[80] Amazon Route 53. Available: `http://aws.amazon.com/route53/`.

[81] Daniel Turull, Markus Hidell, and Peter Sjödin. libNetVirt: the network virtualization library. In *Workshop on Clouds, Networks and Data Centers (ICC'12 WS - CloudNetsDataCenters)*, Ottawa, Canada, June 2012.

[82] Vxdl forum. `http://www.vxdlforum.org`.