Objective FP7-ICT-2009-5-257448/D-5.2

Future Networks

Project 257448

"SAIL – Scalable and Adaptable Internet Solutions"

D-5.2 (D-D.1) Cloud Network Architecture Description

Date of preparation: **11-07-31**
Start date of Project: **10-08-01**
Project Coordinator: **Thomas Edwall**
Ericsson AB

Revision: **1.0**
Duration: **13-01-31**

Document Properties

Document Number:	D-D.1
Document Title:	**Cloud Networking Architecture Description**
Document Responsible:	Paul Murray (HP)
Document Editor:	Paul Murray (HP)
Authors:	Bob Melander (EAB), Victor Souza (EAB), Volker Fusenig (Fraunhofer), Ayush Sharma (Fraunhofer), Mickael Meulle (FT Orange), Dev Audsin (HP), Paul Murray (HP), Suksant Sae Lor (HP), Luis Vaquero (HP), Thomas Begin (INRIA), Paulo Gonçalves (INRIA), Guilherme Koslovski (INRIA), Shubhabrata Roy (INRIA), Wajdi Louati (IT), Marouen Mechtri (IT), Houssem Medhioub (IT), Djamal Zeghlache (IT), Markus Hidell (KTH), Rolf Stadler (KTH), Peter Sjodin (KTH), Daniel Turull (KTH), Fetahi Wuhib (KTH), Pascale Vicat-Blanc (Lyatiss), Dominique Dudkowski (NEC), Jorge Carapinha (PTIN), Marcio Melo (PTIN), Joao Soares (PTIN), Romeu Monteiro (PTIN), Daniel Gillblad (SICS), Rebecca Steinert (SICS), Björn Bjurling (SICS), Björn Levin (SICS), Avi Miron (Technion), Pedro Aranda (TID), Ibrahim Menem (TID)
Target Dissemination Level:	PU
Status of the Document:	Final
Version:	1.0

Production Properties:

Reviewers:	Bengt Ahlgren (SICS), Hannu Flinck (NSN)

Document History:

Revision	Date	Issued by	Description
1.0	2011-07-31	Paul Murray	Final version

Disclaimer:

Document:	FP7-ICT-2009-5-257448-SAIL/D-D.1		
Date:	July 31, 2011	Security:	Public
Status:	Final	Version:	1.0

Abstract:

This document describes the first complete version of the Cloud Networking architecture proposed by SAIL WPD. The concept of cloud networking is introduced in a multi-administrative domain scenario, where network and data centre domains exist and must interact through defined interfaces to provide a service to cloud customers. Cloud networking builds upon two main concepts. The first is integration of the networking resources onto existing data centre based cloud infrastructures. The network resource is represented by defined flash network slices which are dynamic elastic network connections. The second concept is the deployment of computing and storage resources distributed in the network to allow for better end-user experience and lower the dependency on network capacity. The architecture introduces administrative domains, interfaces, logical functions, and a description of inter-domain interactions to provide complete end-to-end services. Management algorithms for user goal translation, fault and resource management are presented, including early results from experimentation and simulations. Security goals are outlined along with a description of proposed approach to security goals and policy based access control. This architecture will be further refined and modified according to results from implementation of a prototype and experimentation.

Keywords:

cloud computing, cloud networking, network virtualisation, infrastructure as a service, cross-domain virtual infrastructure, resource management, cloud network security

Contents

List of Figures

List of Tables

List of Acronyms

4WARD	Architecture and Design for the Future Internet
API	Application Programming Interface
BGP	Border Gateway Protocol
CDMI	Cloud Data Management Interface
CloNe	Cloud Networking
DCP	Distributed Control Plane
DoS	Denial of Service
ECC	Elliptic Curve Cryptography
ETSI	European Telecommunications Standards Institute
FNS	Flash Network Slice
IaaS	Infrastructure as a Service
IP	Internet Protocol
ISP	Internet Service Provider
MOM	Message Oriented Middleware
MPLS	Multiprotocol Label Switching
NetInf	Network of Information
NIC	Network Interface Card
OCCI	Open Cloud Computing Interface
OCNI	Open Cloud Networking Interface
OConS	Open Connectivity Services
OGF	Open Grid Forum
PCI DSS	Payment Card Industry Data Security Standard
PKI	Public Key Infrastructure
QoE	Quality of Experience
QoS	Quality of Service
REST	Representational State Transfer

S3	Simple Storage Service
SAIL	Scalable Adaptive Internet Layering System
SDLC	Software Development Life Cycle
SLA	Service Level Agreement
SNIA	Storage Networking Industry Association
SOA	Service Oriented Architecture
URI	Universal Resource Identifier
UUID	Universally Unique Identifier
VLAN	Virtual Local Area Network
VM	Virtual Machine
VoD	Video on Demand
VPN	Virtual Private Network
VXDL	Virtual private eXecution infrastructure Description Language

1 Introduction

Cloud computing is set to form the bedrock of a fundamental change in the way enterprise IT and consumer services are implemented and delivered. Already cloud computing providers are offering infrastructure-as-a-service, implementing virtualised compute, storage and network as an environment in which to deploy and run applications and services; and new enterprise IT and consumer services implemented in these environments are appearing every day. Paid for according to use, such environments afford service providers the ability to scale far beyond the limits of investment justified by fluctuating and uncertain demand and without impacting user quality of experience.

Today Infrastructure as a Service (IaaS) is an immature and proprietary technology largely focused on data centre provision of compute and storage facilities. The networking aspect of infrastructure as a service and the lack of of network support beyond the data centre are a limiting factor preventing applications and communication services that are sensitive to network characteristics from adopting this approach. The related infrastructure service interfaces need to be standardised in order to facilitate wide and smooth deployment and to provide a consistent access by users to any cloud provider.

The subject of this document is an architecture for Cloud Networking (CloNe). The aim of CloNe is to complete the cloud computing picture by addressing the networking aspect. We introduce the concept of a *flash network slice*, a network resource that can be provisioned and dimensioned on a time scale comparable to existing compute and storage resources. Flash network slices can be used to construct and connect virtual infrastructures spanning providers, creating environments in which to deploy, scale and distribute services across data centres, throughout networks, to end users across the globe.

1.1 Architectural Constraints

The CloNe architecture is grounded in a few basic constraints derived from the nature of networking and the current trend in cloud computing. These are described as follows:

- **Can be realised on currently deployed network technologies** – in recognition that it is not viable to take a revolutionary approach to changing the way existing deployed networks operate we wish to derive an architecture that can be used in existing deployments.
- **Can be realised on network technologies that are expected in the future** – we expect our architecture to encompass new networking technologies that more effectively support dynamic configuration of virtual network resources and to provide a migration path to the adoption of such technologies.
- **Compatible with current approaches to cloud computing** – IaaS offerings already have emerging standards for management of computing and storage resources. To embrace infrastructure as a service as a way to provision a combined networking and computing infrastructure we need to cooperate with these existing components.
- **Can be realised across multiple administrative domains** – we expect network providers to interoperate with each other and with privately owned infrastructures and end users, such

that CloNe infrastructures can be constructed from virtual resources realised in different administrative domains, with technological and operational independence.

1.2 Business Justification

The deployment of infrastructure and applications in a cloud is today largely motivated by cost reduction. Cloud computing features on demand provisioning of resources without upfront commitments; elastic reconfiguration of those resources and pay-per-use business model. Any applications that show a high degree of demand fluctuation will likely yield lower cost when deployed in a cloud, since provisioning is not done for peak demand. Cloud computing can significantly reduce capital expenditure, leaving customer only with the costs of operating the IT infrastructure of their businesses.

Cloud networking proposes the deployment of in network servers (cloud servers) in order to improve end-user experience. In that scenario, servers can be placed close to end-users allowing for lower latency and packet loss ratio avoiding congested links when accessing that server. These servers should be placed in multiple places throughout the network. The number of servers in each one of these of points of presence will be much smaller than a traditional data centre cloud; however, they will be serving a much smaller part of the users as well. Applications that demand interactivity will benefit of such deployment. We can this type of deployment a distributed cloud.

This scenario does not substitute the need for traditional data centre deployed clouds, where massive amounts of servers can be provisioned. The business case of such a distributed cloud leverages upon the enhanced end-user experience for applications where the data centres may be too far away. It is envisioned that a pay-per-use model could be used as well. Pricing may be differentiated though.

The second important aspect is the provisioning of elastic networking connectivity. A possible business model for the elastic networking is a pure extension of the cloud computing business model. Cost reduction will happen since customer will be able to adapt the network service according to the application demand. During low usage periods (e.g., evenings) the customer should be able to dimension its service accordingly and pay only for the needed network capacity.

Moreover, cloud networking goes beyond the above to propose new business cases where different providers cooperate to deploy a given application. The cooperation is made on the basis of mutual deployment of different parts of the customer services. An infrastructure provider may not be able to fully satisfy a user request. It may, however, satisfy part of it and delegate the deployment of the rest to another infrastructure provider. The user does not need to be aware of that, as long as all of its requirements are satisfied. Those requirements may be related to legal aspects, e.g., deployment of application under a certain jurisdiction, or to security level associated to data storage, amongst others.

The concept of delegation has important implications on the business case of the whole solution. First of all, a business to business interface amongst infrastructure providers needs to be established. Infrastructure providers should be able to automatically negotiate and charge each other for provided services. Second, it allows the user to deploy its applications across many providers through one business relationship. Essentially, cloud customers can continue to use their existing business relationships and benefit from the cross provider infrastructure deployment enabled by cloud networking.

Delegation allows for CloNe providers to deploy fewer resources since each of them can rely on remote resources to fulfil its peak demand. Different CloNe providers experience peak demand at different times and delegations may mean a cost-saving opportunity. Clearly, each CloNe provider will have to decide upon the amount of infrastructure to be deployed, considering cost, and the risk of not having other providers to supply resources during periods of high demand.

Finally, delegation will enable the creation of a marketplace where infrastructure providers can negotiate resources amongst each other. Infrastructure providers may decide to establish static sharing agreements. Dynamic agreements will however foster competition and allow for a greater degree of flexibility when deploying cloud services.

1.3 Document Outline

The approach of this document is to describe the architecture of an infrastructure service from a high level of abstraction and then demonstrate how components of the architecture relate to real technologies in order to validate that it conforms to the basic constraints described above.

The high level architecture is presented in Section 2, including a framework for characterising virtual infrastructure, the flash network slice infrastructure resource, the roles involved in interaction with an infrastructure service, interfaces relevant to an infrastructure service, and management functions of an infrastructure service. The delegation concept is introduced in this section.

Section 3 describes the characteristics of a data model for virtual infrastructure including the flash network slice.

Section 4 characterises the information flows of control aspects of the architecture and presents an example design for an infrastructure service that supports control interaction across multiple administrative domains.

Section 5 describes a framework of core management functions that compose an infrastructure service and demonstrates how management algorithms can be implemented in that framework.

Section 6 focuses on realisation of the flash network slice resource in various technologies, from those commonly deployed today to those that may be deployed in the future.

Section 7 describes a security architecture based on accreditation and compliance that can be used in conjunction with the technical architecture.

Section 8 describes how the use cases described in [3] can be implemented in the CloNe architecture.

Section 9 relates our work to work carried out in other projects and Section 10 summaries the current state of CloNe and identifies future work.

CloNe is one of three technical work packages of the Scalable Adaptive Internet Layering System (SAIL) project; the others are Network of Information (NetInf) and Open Connectivity Services (OConS). Each work package describes an architecture for its respective system and the relationship among these is covered in [4].

The SAIL project addresses four themes across the work packages: *security*, *management*, *inter-provider issues*, and *prototyping*. These themes are addressed throughout the CloNe architecture. The security theme is addressed in the security architecture described in Section 7 and the authentication interfaces outlined in the high level architecture in Section 2 and the control view in Section 4. Management is addressed in the infrastructure service functions of the high level architecture and in Section 5. Inter-provider issues relate to the cross-domain aspects of the architecture including control in Section 4 and reference resolution in Section 3. Prototyping activities are at an early stage in CloNe and are beyond the scope of this document.

2 High Level Architecture

The CloNe high level architecture consists of four parts: the *three layer model*; a set of roles; a set of interfaces by which the roles interact; and a set of management functions in which these roles participate. The three layer model is a framework for characterising virtual infrastructure relative to three different view-points. The roles, interfaces and management functions are characterised by reference to the three layer model. This section describes each of these parts. In addition the Flash Network Slice (FNS), a new component of virtual infrastructures introduced by CloNe, is defined in the context of the architecture.

2.1 Three Layer Model

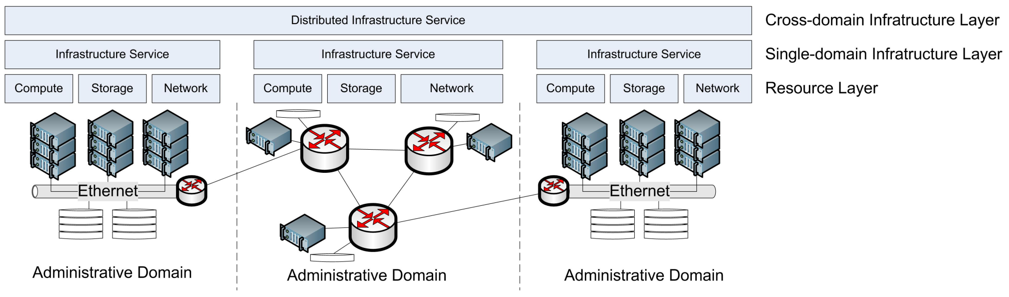

Figure 2.1: Three Layer Model

Figure 2.1 is a visual representation of the view-points used to characterise the components of the CloNe architecture. An administrative domain is a collection of physical or virtual equipment that is under the management of an administrative authority. As Figure 2.1 shows, examples of administrative domains in the context of CloNe are wide area networks and data centres. Virtual infrastructure is within an administrative domains and a single infrastructure may span multiple domains. Administrative domains are depicted at the bottom of Figure 2.1. The three layers above the administrative domains represent three different views of virtual infrastructure. The three layers include *resource*, *single-domain infrastructure*, and *cross-domain infrastructure*. Later we describe the roles, interfaces and functions in relation to each of these views.

The way authority is segregated over administrative domains influences the management of virtual infrastructure and the construction of the three layer model.

2.1.1 Resources

A virtual resource is an abstract representation of a component of a virtual infrastructure such as a virtual machine or a volume on a block device. A virtual resource resides within the boundaries of a single administrative domain. The resource layer includes compute resources, storage resources, and network resources as virtual entities that are generally managed by different sub-systems. Resources can be identified (or named), they have certain properties and status and may have links to other resources.

Properties, status and links are not the same. A property is externally determined (i.e. it is an attribute set from the outside such as memory size for a virtual machine or address space for a subnet), a status is internally determined (i.e. it reflects a condition of the virtual resource: a life cycle stage, an error condition or an attribute derived from its circumstance), a link describes a relationship to another virtual resource and is interpreted relative to the other resource.

It is relevant to notice that CloNe proposes resources to be at the bottom of both data centres and operators networks. CloNe proposes computing and storage resources to be deployed in the operators networks (e.g., wide area networks). Similar resources (e.g., a virtual machine) may have a different property set depending on the administrative domain where they reside. For example, the network location of a virtual machine is a property that is relevant in the wide area network but not as much in the data centre. Regardless, from a management perspective, both are simply resources.

A virtual resource can be created, managed and destroyed. These control actions are typically performed by a subsystem such as the management interface on a virtual machine hypervisor or a storage device manager such as a storage array control system. The act of creating a virtual resource will typically consume some physical resource (e.g. disk space) or a logical resource (e.g. an Internet Protocol (IP) address or bandwidth). Destroying a virtual resource will free these physical or logical resources.

Virtual resources are often connected to other virtual resources: a virtual machine may be connected to a virtual network or a storage device, establishing a relationship. The nature of this relationship can determine if the virtual resource can be correctly established in its own right or if it depends on the existence of the related virtual resource, its properties or its status. We assume interaction between the control mechanisms of different virtual resources within a single administrative domain where it is necessary to interpret the condition of a link. As an example a virtual machine manager may have to interact with a storage device to connect a virtual machine to a network attached volume. It may also communicate with a network management interface to add a virtual machine to a virtual network.

A virtual resource can be managed by a single administrative domain, but may have links with virtual resources in other administrative domains. This is particularly true of network connectivity as it is the means of interacting beyond the boundary of one provider. We assume interaction between control mechanisms of different resources in different administrative domains where it is necessary to interpret the condition of a link or to establish a link.

2.1.2 Single-Domain Infrastructure

We consider a single-domain infrastructure to be a number of virtual resources managed collectively within a single administrative domain. The links among these virtual resources determine the topology of the infrastructure and constrain their combined management behaviour.

A single-domain infrastructure is managed by a single administrative authority that has management rights over the underlying equipment and virtualisation technology. As a consequence, within a single domain the administrative authority has full knowledge about the available resources and virtualisation capabilities at any time. A single-domain infrastructure can be created, updated, managed and destroyed.

At this layer the mapping between the single-domain 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 Virtual Machine (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 Virtual Private Network (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.

2.1.3 Cross-Domain Infrastructure

We consider a cross-domain infrastructure to be a number of virtual resources managed collectively across multiple administrative domains. A cross-domain infrastructure can be partitioned into multiple single-domain infrastructures. A single-domain infrastructure may contain virtual resources that have links with virtual resources in other single-domain infrastructures, thus connecting the virtual infrastructures and determining the topology of the cross-domain infrastructure.

A cross-domain infrastructure is managed by multiple administrative authorities. In contrast to a single-domain infrastructure, the state of underlying equipment and virtualisation capabilities are likely not fully shared beyond domain boundaries. In this type of infrastructure, specific interfaces via which resource virtualisation is negotiated are necessary. Via such interfaces, the authorities of administrative domains may exchange limited information that they are willing to share about their domain in order to facilitate the optimization of cross-domain virtualisation. A cross-domain infrastructure can be created, updated, managed and destroyed.

Decomposing the requested virtual infrastructure into administrative domains can be performed at this level based on the capabilities of the administrative domains and their interconnection. Properties and links of the virtual resources and properties of the requested virtual infrastructure as a collection will obviously influence the decomposition process.

2.2 Flash Network Slice

The FNS is a new resource that is introduced by CloNe. The purpose of a FNS is to more fully address network capabilities in the IaaS paradigm according to CloNe requirements. A FNS has the following properties:

- **Network resource**: it is a resource providing network communication capability.
- **Links**: it can be attached to other resources by links, for example a VM may be attached to a FNS or two FNSs may be attached to each other. A link may span administrative domains.
- **Space of interconnected links**: it implements message forwarding between links.
- **Quality of service**: it has quality of service properties associated with its communication capability between links.
- **Single administrative domain**: it is constructed and managed within a single administrative domain (this conforms to the definition of a resource given above).
- **Set up time**: it is established automatically and in a time frame comparable with existing virtual infrastructure resources such as VMs.

The FNS provides a communication capability between resources that are linked to it. Where two FNSs are linked to each other they provide a communication capability that spans the two slices. As links between FNSs are allowed to span domains, this provides a communication capability that is fundamental to the establishment of infrastructures across domains.

2.3 Roles

The use cases that guide the definition of the CloNe architecture are described in [3]. Here we define the common set of roles that are representative of those found in the use cases:

- **Administrator** has administrative authority over underlying virtual or physical equipment (the administrative domain) used to implement resources. The administrator uses management systems to configure and manage resources within the administrative domain.
- **Infrastructure Service User** accesses an infrastructure service in order to obtain, examine, modify and destroy resources.
- **Infrastructure Service Provider** offers an infrastructure service that may be used by an *infrastructure service user* to obtain, examine, modify and destroy resources.

The roles are acted out by various organisations involved in specific use cases. Typically the use cases involve data centre operators, network operators or agents reselling access to virtual resources to implement an infrastructure service. Use cases involve a user that obtains access to the infrastructure provided by the infrastructure service: such as an enterprise, an application service provider or a communication service provider. Although most scenarios involve a user of the application or communication service created within a virtual infrastructure, these types of user do not interact with an infrastructure service in that capacity. Roles may be added to represent these actors in specific use cases, and other actors as required, but they do not form part of the common set we describe in the high level architecture.

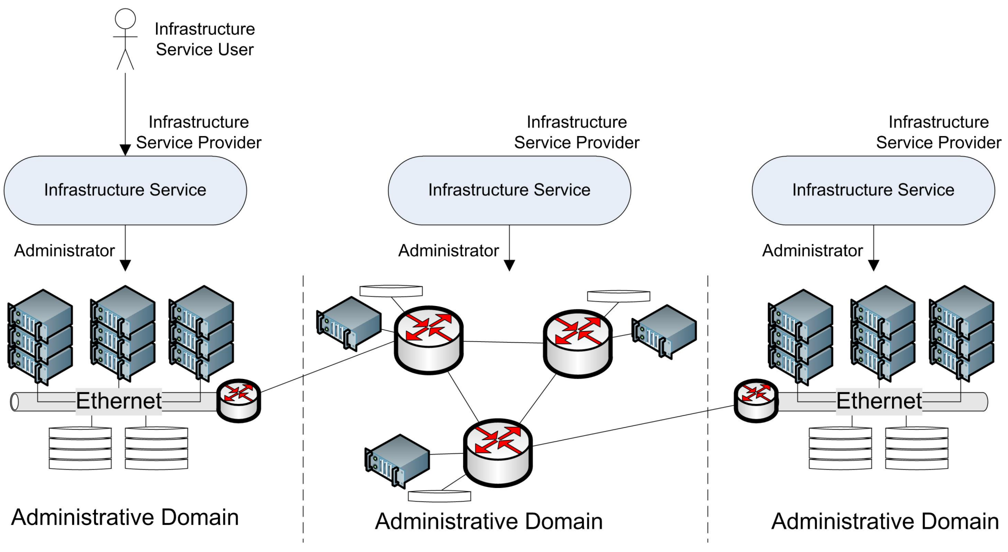

Figure 2.2: Roles in a Simple Arrangement

Figure 2.2 depicts a simple arrangement of infrastructure services to demonstrate where these roles fit in an infrastructure service. The infrastructure services in the figure have the *administrator* role, meaning they have access to their own administrative domain and can manage individual virtual resources directly. The administrator role performs management of individual resources, placing it and the management systems it uses at the resource layer of the three layer model. The infrastructure service in this figure also adopts the *infrastructure service provider* role to offer service to an *infrastructure service user*. The service coordinates management of the virtual resources within its own administrative domain, placing the infrastructure service provider and its service at the single-domain infrastructure layer of the three layer model in this case. The infrastructure service user at the top of the figure represents an organisation that makes use of virtual infrastructure. The interaction between the user and provider represents delegation; the provider is responsible for implementing and managing the virtual infrastructure on behalf of the user.

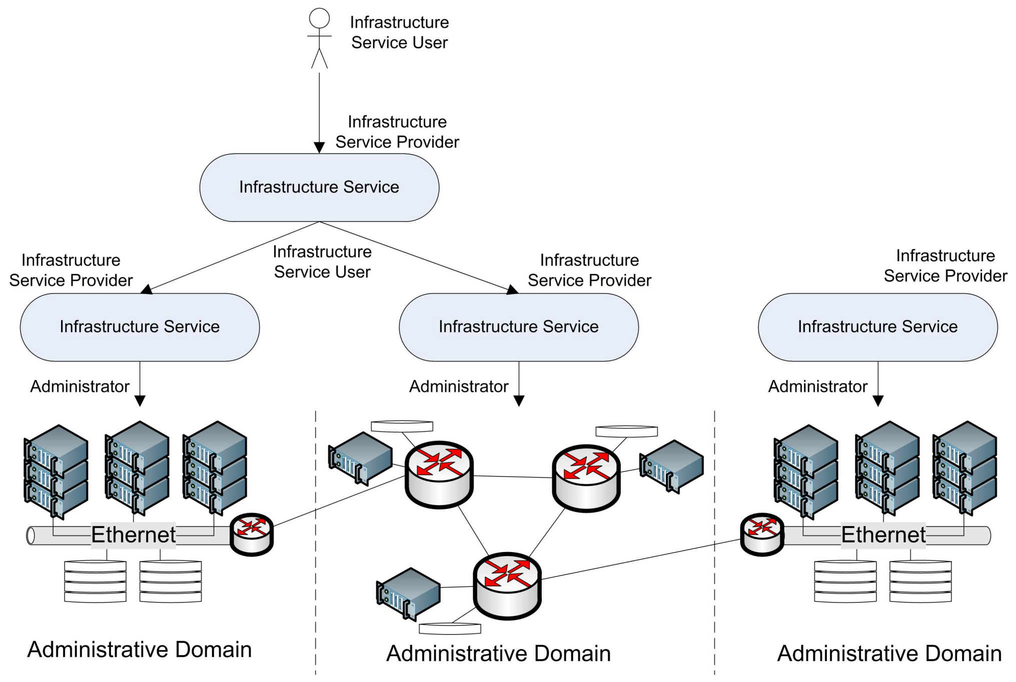

Figure 2.3: Roles in a Hierarchical Arrangement

Figure 2.3 depicts a more complex arrangement including a delegation hierarchy among infrastructure services. In this case the topmost infrastructure service adopts both the provider and user roles and is responsible for mapping the original user's infrastructure requirements into parts that are implemented by the lower providers. This coordination of resources across multiple infrastructure services is logically equivalent to a single instance of a service implementing the infrastructure across domains (although the user may be aware of the final providers) and places the provider and service in this case at the cross-domain infrastructure layer. This service instance creates a delegation chain and it is the services with the administrator role that are responsible for implementing and managing the virtual infrastructure on behalf of the original user; the intermediary services track the delegation.

As is shown, the infrastructure services can be placed at either the single-domain or cross-domain infrastructure layer. The implication of each is that different types of behaviour are implemented depending on what roles they adopt. It is possible for an implementation of an infrastructure service to have both the infrastructure service user and administrator roles, implying it can both delegate and directly implement infrastructure, in which case it spans both layers. This can be used to implement a peering arrangement as shown in Figure 2.4.

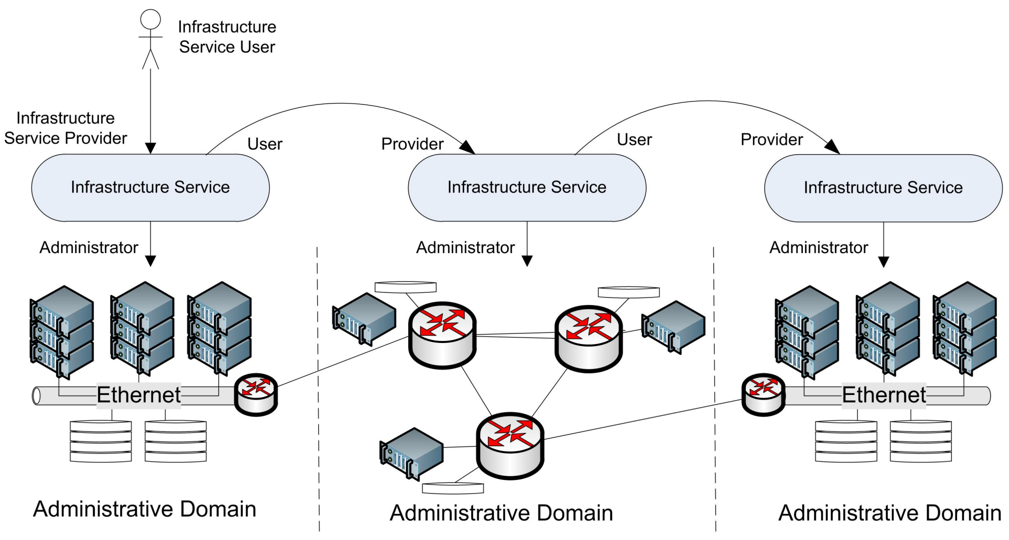

Figure 2.4: Roles in a Peering Arrangement

2.4 Interfaces

Three interfaces are identified between the different roles of the architecture. They are the *resource administration interface*, the *distributed control plane* and the *infrastructure service interface.* These are depicted in Figure 2.5 and described below.

2.4.1 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 [5] virtual machine manager interface, a storage device controller or a network administration interface.

Basically, these 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

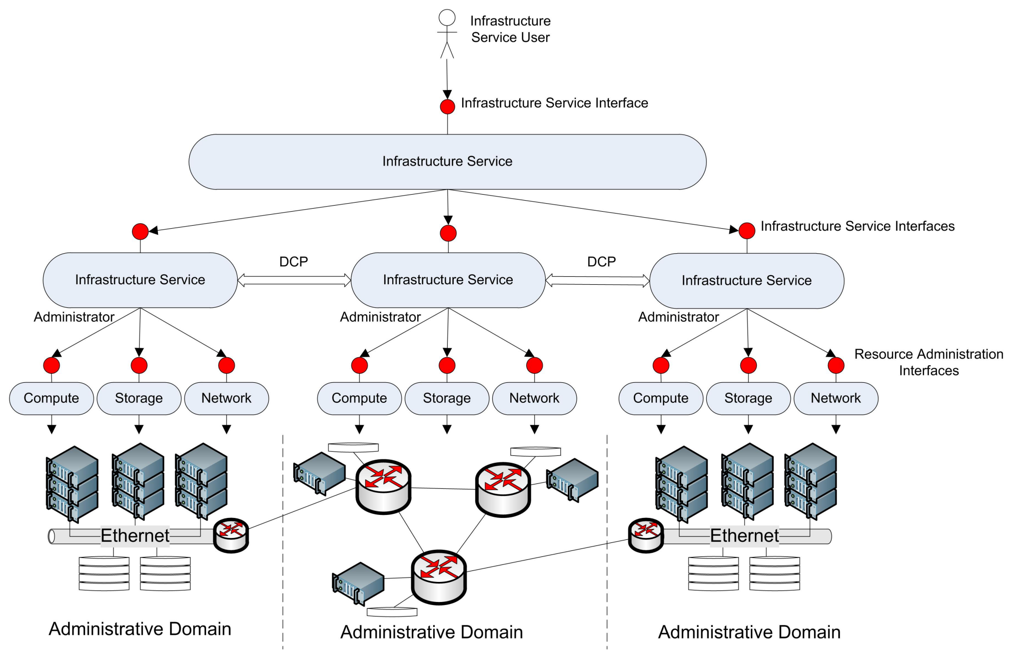

Figure 2.5: Interfaces

- 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. These two types of interfaces need to blend and interoperate.

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 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 may be supported, as well as fault monitoring, and redundancy. 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 Distributed Control Plane

The Distributed Control Plane (DCP) describes a category of protocols, interfaces and control operations that enable two or more *infrastructure service providers* to interact and exchange cross-administrative domain information. A more precise definition of the protocols and interfaces that constitute the DCP is the subject of future work. Here we give an informal description of some interactions that are expected to occur between providers.

- **Reference resolution**: the process of converting an abstract representation of remote information to the actual information. As an example, if a network link is to be established, information about the remote end of the link may be required. That information may be represented by an abstract reference that can be resolved through the DCP to obtain the actual information. Reference resolution is described in more detail in Section 3.
- **Notification**: asynchronous information exchange, including publish-subscribe and asynchronous callback protocols. This type of information exchange is used to decouple the request for information from the response and is useful for distributed coordination. As an example, the establishment of a network link across domains may require cooperation between two providers that are operating asynchronously. One may request information from the other before the other is ready to supply the information. The notification service provides a means to transfer the information when it is available. As another example the information may rarely occur and is a trigger for processing (as opposed to being requested by processing). This is the case for fault-notification.
- **Distributed information sharing**: a distributed information service may provide a global view of infrastructure status information. The view may be maintained through a distributed protocol across providers allowing inspected by the providers.

The DCP operates at the cross-domain infrastructure layer and is generally concerned with distributed coordination and global information access. Communication between domains 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.

2.4.3 Infrastructure Service

The infrastructure service is a fundamental part of the CloNe architecture. The infrastructure service is the set of interfaces that enable the creation, monitoring and management of virtual infrastructures provided by the *infrastructure service provider* role and accessed by the *infrastructure service user* role.

We assume that user requests to the infrastructure service will be performed using a high level description language. Those requests need to be broken down into low level actions inside the control plane for allocation of underlying resources within different domains. A high level description of the user requirements should yield a simpler way for the user to utilise resources that may be largely distributed. A good example of that is the location of the resources being utilised. The client interface should make it easy for the user to utilise highly distributed servers without having to bother about their specific topological location.

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 translation of the description language into control actions has to work in a distributed setting under the uncertainty that other parts of the cloud only partly fulfil their contracts.

For configuration and monitoring of services, cloud network management functions provide the functionality required to manage the cloud's virtual resources and should also facilitate management of legacy networks. The functions can be viewed as higher-layer management entities that address cloud network specific requirements, utilizing available legacy management capabilities when necessary. For efficient and distributed cross-communication and information exchange between management functions, collaboration is enabled through two interfaces - a controller interface and a collaboration interface (see Section 5.2).

The infrastructure service requires the use of well-defined security modules, in order to satisfy its security requirements, provided through user/operator/legal requirements, i.e., authentication, auditing, confidentiality, integrity and assurance, besides others. The 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 7.2.

2.5 Service Request Management

The cloud network management consists of three high-level management functions: *goal translation* (GT), transforming business, technical and security goals into resource configuration objectives; *fault management* (FM), providing status updates of monitored resources, measurements and models, as well as detecting faults, anomalies, and changes in performance; and *resource management* (RM), responsible for provisioning, allocation, reconfiguration, optimisation of resources and performance. Approaches and algorithms implementing management functionality are described in Section 5.

The management functions operate within infrastructure service providers and contribute in managing the administrative domain. Functional blocks within a management function implement a certain functionality using a set of management algorithms. The management algorithms need to be efficient with respect to their overhead, scalable with respect to the number of managed entities and adaptive as well as robust to all types of changes in the managed cloud. In order to achieve these properties, the algorithms operate in a decentralised manner, ensuring scalability, efficiency and reliability for the size of clouds envisioned. Decentralised collaboration between the management functions allows for administration, configuration and monitoring of individual FNSs as well as the cloud network. Through the DCP and management function interfaces (see section 2.4), the management functions exchange information for decentralised, adaptive and efficient management in both single-domains and cross-domains.

3 Data Model

CloNe adopts a model-based approach to describing and managing virtual infrastructure. The data model view describes the representation of virtual infrastructure and the format for its exchange among system components. A consistent representation is particularly important in this architecture as different system components are likely to be implemented by different parties, so a common understanding of the model and the data format used in exchange between system components is critical.

3.1 Data Model Principles

This section introduces how the description of the application, as provided by the user, is dealt with across the CloNe architecture and what are the governing principles that rule this process. The architecture herein proposed is heavily dependent on a **delegation** process by which different actors perform tasks that have been delegated to them by other actors in possession of a less accurate knowledge or lacking some required information to perform the task at hand. Obviously, the roles played by these actors can be combined so that a single individual/institution can play them as needed.

3.1.1 Delegation

Information describing a virtual infrastructure is passed between an infrastructure service user and an infrastructure service provider through the infrastructure service interface. The user describes the desired virtual infrastructure to the provider and the provider reports on its status to the user. Such descriptive information may be in an abstract form with the provider taking responsibility for determining implementation detail that is hidden from the user. This information exchange represents the act of delegation described in more detail in Section 4.

The presence of a common data model (semantically rich enough for expressing the required goals) and a common mechanism for labelling the entities in the model (so that information can be fed backwards once a delegated operation has been materialized) enables this information exchange.

An infrastructure service can choose to repeat the delegation process, possibly across multiple other providers, until the required infrastructure is implemented within administrative domains. This repeated delegation presents a combination of model transformation and information hiding requirements for the data model that are inherent in the delegation approach.

3.1.2 Model Transformation

Once the user specifies a set of elements to be deployed and how they are connected, the infrastructure service provider labels each element with a unique name. The infrastructure service provider has some information that remains unknown for the user (e.g. underlying infrastructure details or topology of the virtual infrastructure, i.e. administrative domains and contract terms with each of these ones).

3.1.2.1 Abstraction Layers

This abstraction makes life for users easier, they delegate complex tasks to experts who can also benefit from economies of scale in their individual contracts with other infrastructure service providers (if they themselves delegate) or iron vendors (if they also play the role of administrators). For instance, infrastructure service providers will not provide complete visibility into the nature of the underlying physical infrastructure, its virtualisation, or other service implementation details. It will only make a selected set of service capabilities known to the user. We refer to this as information hiding.

On the other hand, users may not really care about the low level implementation and may specify their requests either in the form of a high level description (bubbles and links model) for their services to run, or may specify a more complex scenario in which they want to be in control of the whole networking topology connecting the VMs comprising their service in the cloud (possibly across federated data centres).

3.1.2.2 Model Refinement

This need for abstraction and hiding the underlying complexity to users is done at every level of the proposed architecture. Thus, the data model is a live element that evolves from a very general and vague description (users with little knowledge on the complexity underneath) to a detailed low level description of the components that are needed and their configuration.

The data model, thus, becomes the vehicle conveying the information that needs to be exported at every level (the *goals* for deploying users' requests and the monitoring data fed back to them) and the one that needs to be refined or incorporated (matching in Figure 3.1) in the light of the newly available information when a lower level (in the chain of delegations) actor receives the data model. For instance, let us assume that an administrative domain has been delegated to deploy a VM from an independent infrastructure service provider. The infrastructure service provider lacks any knowledge on the network topology of the administrative domain or the specifics of racks, hypervisors, etc. At a higher level, an infrastructure service provider that delegates may hide details about the infrastructure service providers it uses (looking just as another normal infrastructure service provider to the user). These details contain, for instance, information such as price for deploying a VM, price per stored MB and QoS (e.g. time for a packet to traverse the diameter of the network). This process of *resource management* (matching high level goals specified by a user into something that can be materialized in the light of the, now, available information the user was lacking) and goal refinement (when the request is delegated to other providers of a domain, the goals need to be changed so that they reflect the available information) is essential in taking the user's request to a concrete ground.

Infrastructure service providers may decide to deploy all their requests on a given domain as long as this complies with the users' goals.

3.1.2.3 Model Splitting

Figure 3.1 illustrates a case in which a network is required to span two domains that share a border in the form of connected edge nodes. These nodes need to be configured to interconnect the virtual networks in each domain. As is shown in Figure 3.1, the infrastructure service providers receive models of the network resources that they will implement in their respective domains. It is possible that the configuration details for the edge nodes cannot be completed without information from the neighbouring domain. For example, they may need to exchange ports, IP address or tags depending on the nature of the connection. They may also need to agree which edge nodes to use. It is likely that this is information that they will not be willing to share with anyone other than their neighbouring domain. This implies the information will be obtained by interaction between

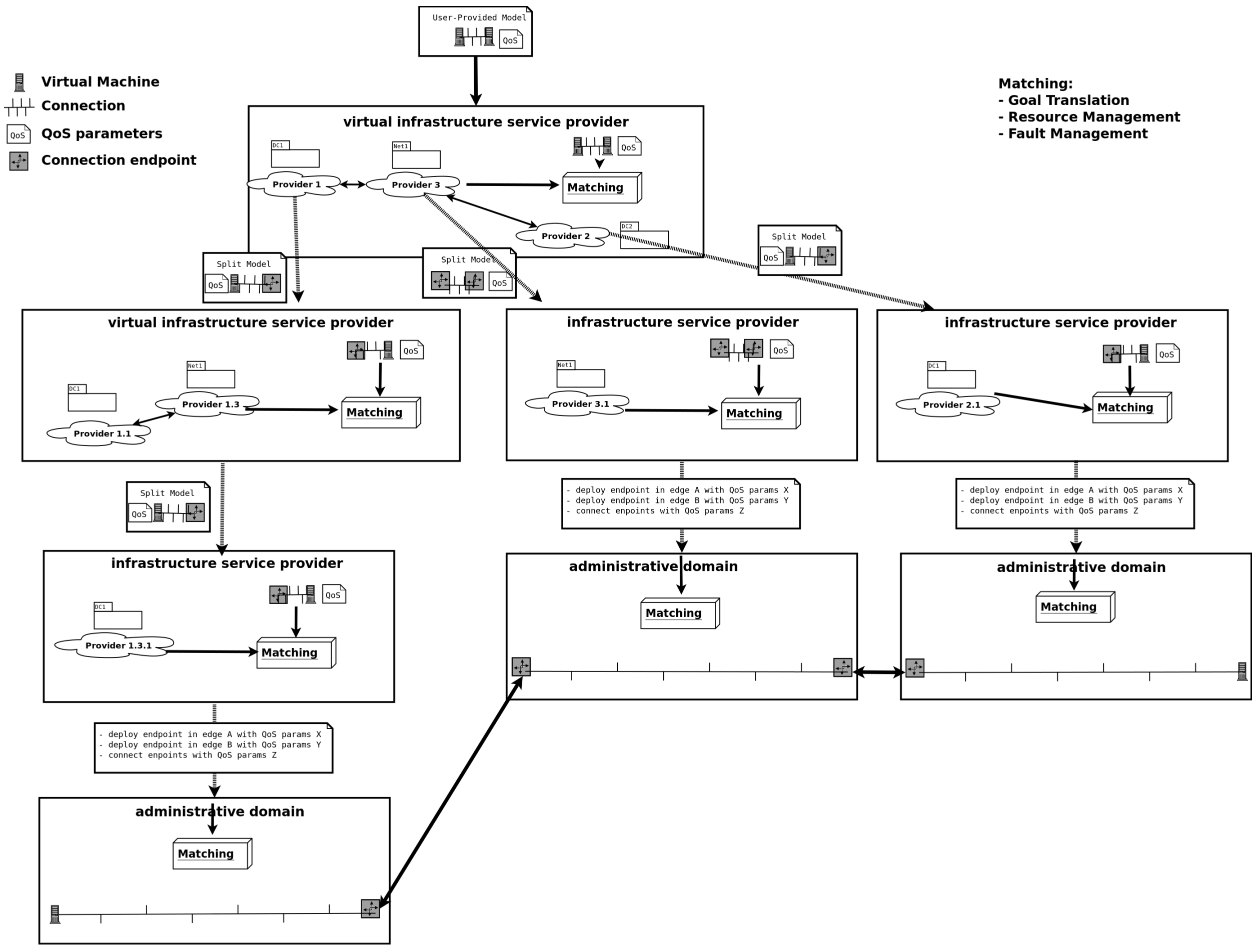

Figure 3.1: A Possible Approach for Information Hiding and Goal Translation.

the lower service providers or in a protocol between the edge nodes - not by interaction with the higher service provider. It also implies that a model will contain information that relates to entities outside the model - such as the neighbouring service provider, a name to identify information held by the neighbouring service provider, or an abstract reference.

3.1.2.4 Dynamic Reference Resolution

Model refinement and splitting has a direct implication over the way things are referenced in the data model. For instance, when a user specifies she desires a VM in the UK and another one in France, she is (likely) indirectly generating a split of her request across multiple administrative domains. When her request is split and refined, the VM in the UK is referring to a VM in France whose final name (e.g. its host name) would only be known at deployment time.

The distributed naming expressed at the last paragraph of the section above (further detailed below in Section 3.2.2) is a way to keep these pointers consistent (a reference points to a unique resource or resource attribute). Once a VM has been allocated, its hostname becomes known and, therefore, the reference can be resolved by back propagating the reference to the required entities in the system (even the user if needed).

Generally speaking a reference will take the form of a protocol+container+resource. A universally accepted example is the Universal Resource Identifier (URI):

http://www.sail-project.eu/resources/vms/vm1/hostname/vm1.sail-project.eu.

Also, the naming scheme under development in SAIL's NetInf work package is being considered for this crucial task.

This simple example is revealing the need for aligning naming and reference resolution so that once a reference is dynamically resolved, it gets delivered back to the interested parties (e.g. VM in the UK gets to know the host name for the VM in the UK). Of course, these are more related to implementation matters than to an architectural definition, but they are issues worth pointing out when tackling the design of the system sketched by this architectural document.

3.2 Basic Model Concept

The architectural model should have a formalised representation of the previous description on how the elements in the model will be dealt with. Thus, a first step towards a valid data model is to have a clear view of the entities in the system and how they are related to each other. The main two elements in the system are, therefore, the resources that need to be deployed or configured at each level and the relationships, links among them.

The basic abstractions included in these data models are the network, the compute and the storage resources (children of the resource class).These elements are, in turn, related to each other by the relationships illustrated in Figure 3.2.

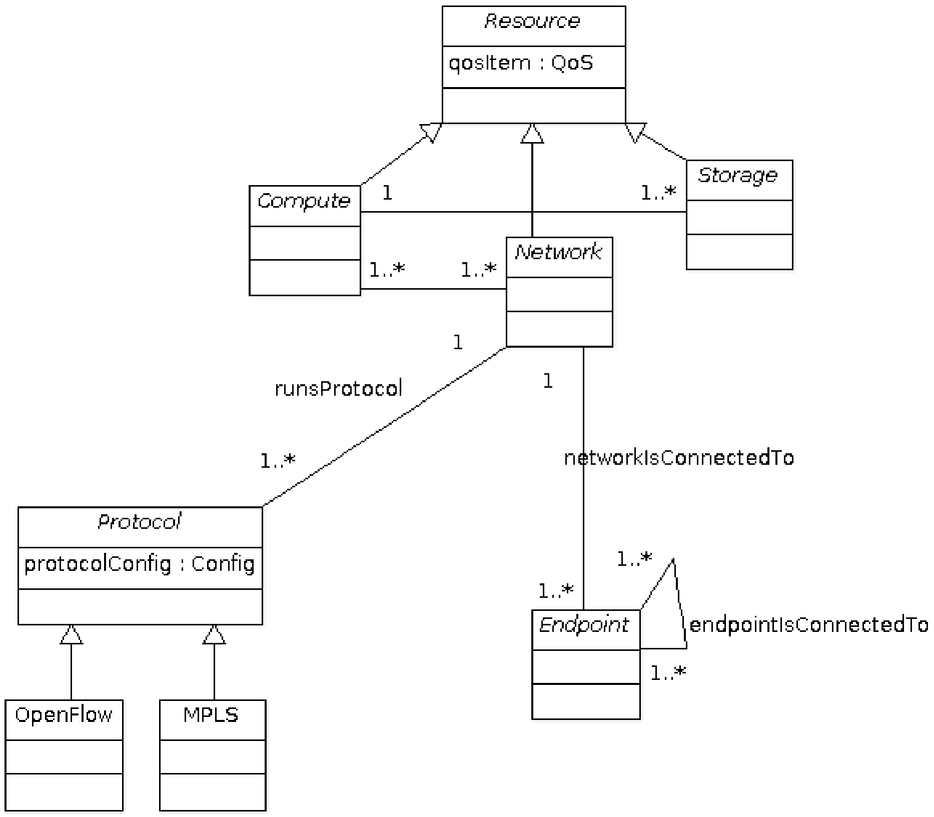

Figure 3.2: A Simple Data Model.

There are various data models and related APIs trying to make their way in the standardisation process of different organisations. Open Cloud Computing Interface (OCCI) is one such standard to-be that is implemented by OpenNebula [6, 7]. OpenStack's network API is another example, but in this case the product of an open source development process. OpenStack's network API and OpenNebula's OCCI both use a simplistic network scenario, using a single network or VLAN networking without routing or bridging (two networks cannot be directly linked/bridged together; a compute resource is needed for them to be bridged). More recent proposals (including future design proposals for OpenStack) start to analyse more sophisticated data models to go beyond some of the limitations present in currently available specifications and APIs [8, 9].

However, all employ network types that are used to represent a network element in the context of a (logical) data centre. This is well enough if the focus of the model is end-user oriented (as in providing a descriptive language such as Virtual private eXecution infrastructure Description Language (VXDL)[8] to allow users to describe their need for specific constraints, rather than expressing the way their need will be instantiated on an actual network of resources). An operator-oriented model might require to deal with the peculiarities of the data centre to network operator or network operator to network operator borders beyond Border Gateway Protocol (BGP) and the like. There is not a means to deal with both, users and operators (data centre and network) peculiarities at same time.

In the following section we will show how more sophisticated mechanisms are needed in order to achieve a fully functional federation of heterogeneous infrastructures (not just data centres). This federation should be capable of preserving infrastructure operational independence (e.g. technological heterogeneity and hidden management mechanisms), while coping with highly abstract user requests to meet a series of application performance, quality of service, cost, etc. goals.

3.2.1 A Possible Embodiment of the Data Model

The first ingredient for understanding how the data model is going to be treated and refined in SAIL is having a high level view of the resources involved in the application. The reader is advised this is just an introductory data model that will be further refined in later sections. A detailed specification of the data model is beyond the purpose of this architectural document. The two major components of this model are the Resources (mainly network, compute and storage resources) and the Relationships among them, also referred to as links in some available data models for describing virtual infrastructures (e.g. OVF's, OCCI's) and topologies (e.g. NDL's), or both such as VXDL's. Figure 3.3 shows these two major elements and how the rest of the elements in the description are organized around them. For instance, VXDL's and OCCI's data models are totally compatible in the sense that network, compute and storage (the main entities in OCCI's) can be considered as children of the VXDL's resource. OCCI's links are directly translatable into VXDL links and, therefore, both model seem to be highly compatible. More detailed research on the appropriateness of any of the available languages is beyond the scope of this document.

The point here is that all the available models can be viewed as a graph of connected resources whose attributes and completeness is going to be refined in the delegation chain described above. The model is the portrayer of user's goals.

3.2.2 Unique Universal Naming in a Distributed Cloud

Being able to delegate a request and let other providers materialize it implies two major things: there is a trust chain across providers; and this trust is maintained even though providers intendedly hide the complexities of their operation (see below) to the users of the services they expose. While the former is highly static and typically done on the basis of written and formal negotiation with different providers, the latter is highly dynamic and needs to be resolved at runtime.

Resolving this naming of entities across federated clouds was easily solved in previous approaches by having a central naming entity (e.g. [10]). A site identifier was followed by the user account and the specific name the user wanted to expose for that given requests. That was the root name for

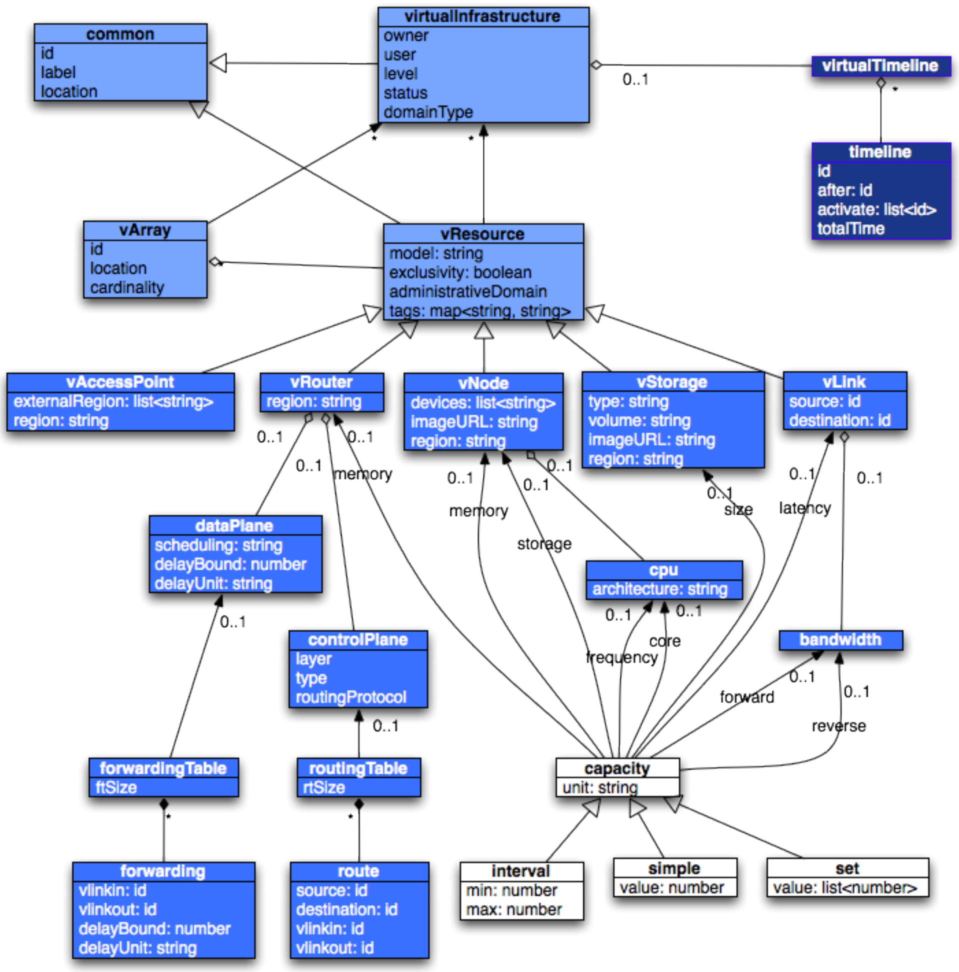

Figure 3.3: Example of Elements Considered in a Virtual Infrastructure + Virtual Topology Description Data Model. Taken from `http://vxdlforum.org`

a request. After that, the name of a given resource could be obtained by appending new branches of the tree as needed (e.g. vms/vm1/cpus/cpu1/ etc.). However, the delegation model herein proposed conveys the federation of resources across non-coordinated domains and service providers and no single entry point for end user requests. Moreover, the presence of virtual infrastructure service providers and the fact that a given virtual infrastructure provider can be, in turn, relying on another virtual infrastructure provider makes naming a bit trickier.

The three most straightforward approaches to distributed naming are, arguably, encapsulation or mapping (see Figure 3.4 for details):

- Encapsulation implies appending length-fixed identifiers and directing the request to the appropriate entity (which is supposed to be capable of understanding and handling it).
- Mapping involves letting service user specify their own names while mapping them locally to unique internal names. This can be done by every player involved in the trust/delegation chain.
- Universally Unique Identifier (UUID) for naming mechanisms. This does not guarantee collisions will not take place, but given the nature of UUIDs their probability of occurrence is kept reasonably low.

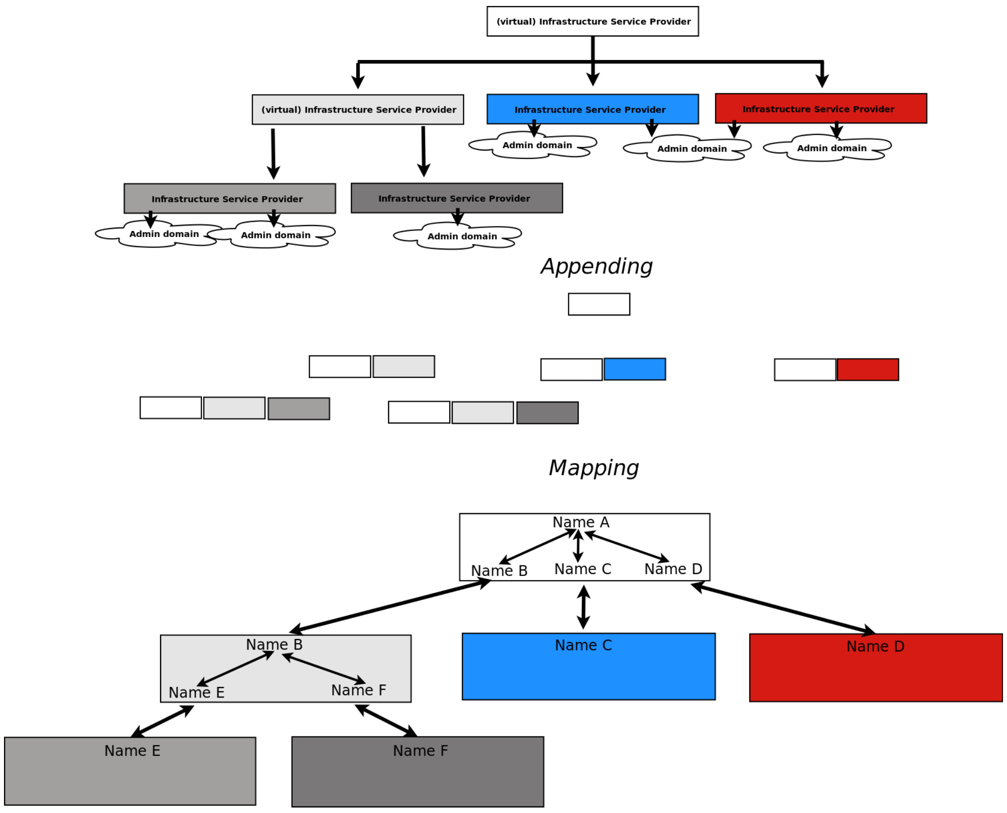
(virtual) Infrastructure Service Provider
(virtual) Infrastructure Service Provider
Infrastructure Service Provider
Infrastructure Service Provider
Infrastructure Service Provider
Infrastructure Service Provider
Admin domain
Appending
Mapping
Name A
Name B
Name C
Name D
Name E
Name F

Figure 3.4: Two Possible Approaches for Distributed and Uncoordinated Naming.

3.3 Network Model

Data centres have experienced a shift towards exposing their capabilities as a service (i.e., IaaS cloud computing). This has boosted innovation by reducing costs of ownership and management, creating new business models where a party can lease out slices of servers on demand and pay essentially for what they use (pay-as-you-go), etc. SAIL aims at making the same come true for networking, where many applications would greatly benefit from in-network functionality beyond the basic connectivity which is offered by most Internet Service Providers (ISPs) as of today.

For instance, in BGP a single route must be chosen at each router, which forces the ISP to announce the same route to every customer. Customers have different needs and different ways to express these: e.g. low latency path vs. forbidden countries in the route. Today's networks do not allow for different routes by giving the customer control over route selection (even in the simplest coarse-grained selection, such as "high bandwidth. For instance, real time applications (e.g. online games) can greatly benefit from more efficient distribution or on-path processing, which currently fall beyond the scope of most ISPs offer.

A lot of attention has been paid to dynamic device network configuration management based on templates [11, 12, 13] and exemplified in BGP configuration or realizing network-wide inter-domain routing policies. Also, a handful of other tools aiming at analyzing the correctness of current network configurations have been reported [14, 15]. Router vendors have also proposed their own network automation frameworks. A major limitation of these automation frameworks is that they remain device-centric and are mostly used to handle local management.

Overlay networks constitute a workaround for these limitations enabling custom functionality with minimal intrusiveness in the network. Multiple virtual networks can coexist on a shared physical substrate through the use of virtualisation. This vision has been partially realized by many undergoing research and pre-production attempts leveraging on virtualisation technologies for a more efficient usage of the underlying resources (e.g. virtual routers, shared NICs in the hypervisors, etc.) [16, 17, 18]. However, most of the prior state of the art has focused on a model in which the network consisted on a series of virtual routers connected via virtual links or basic connectivity [18]. This approach has been deemed to be analogous to the IaaS world, in which the underlying physical resources are split (sliced) so that the operational and cost efficiency are both increased by means of multi tenancy. It misses, however, the on-demand, dynamic, SLA-based and abstracted nature underlying cloud services. After all, one may argue that managing a virtual network is similar to managing a physical network and this falls away from the cloud philosophy: users are supposed to enforce traffic engineering to optimally use their available virtual link bandwidth, "the customer must be able to cope with failure (e.g., by providing redundancy)“ [19, 18]. As mentioned by Keller et al., this conveys inconveniences for both, the application and the infrastructure provider.

3.3.1 Single Router (Node) Abstraction

In order to cope with the limitations of fully virtualised networks specification by application providers while avoiding falling into an over simplistic model basic connectivity functions (as offered by most ISPs today), new data models are required that enable more abstract requests to be expressed by users (e.g. application providers), while still being capable of refining it for more detailed (virtual) network infrastructures are required. The model we use to represent a FNS, the virtual resource we use to provide networking, is based on the *single router abstraction*.

In a first step, user may care about the quality of service parameters for connecting two VMs or storage resources in different network locations (possibly located across several network operators). Most advanced application providers may also deal with specific policies and how packets are handled. The ability to customize path selection within a single domain is another of the advantages of the single router abstraction. In the delegation approach outlined in the previous section, this delegation implies that a given domain may need to pass some of these policies down to the underlying domains and make sure that the aggregation of paths in several domains still comply with the user's specified requirements. In Figure 3.5 one can observe how this delegation approach works in the single domain abstraction. Domain 1 is deploying several VMs in several other domains (since it does not have any actual resources of its own). Domain 1 users see the whole network as a single router to which their VMs can be linked. Domain 1 is hiding the underlying complexity and the information provided by the underlying domains with regards to their neighbours. Domain 1 makes the decision to split the request and deploy the VMs in two different domains. Based on the information exported by domain 2 and 3 about their connectivity (e.g. Domain 1: can connect to domain 2 via Multiprotocol Label Switching (MPLS) or via L3 VPNs at endpoints A, and B with different UUIDs), Domain 1 informs the underlying domains about their need to talk and negotiate in order to get the chosen link (e.g. MPLS connection) ready. Please note that a single switch abstraction is also applicable here, depending on the connectivity level requested by the end user (L2, L3, flowcontrols). This is why we will refer to the single node abstraction as a general case of the single router abstraction introduced by this example.

In this scenario, the concept of the endpoint becomes prominent (represented with the darker blue cylinders in Figure 3.5). And endpoint is nothing else than the concept of a router interface taken to a higher abstraction level. It can be just another resource of the cloud (a VM or physical router) that needs to be properly configured (like, for instance, Ciscos' routers via its IOS commands per interface and protocol). The data model to be used needs to deal with the abstraction of the endpoint (single router interface) and its configuration protocols. Obviously, the specific configuration may be hidden from the user domain and some type of negotiation may take place between two neighbouring endpoints so as to agree in some parameters (for instance, the transmission rate). Roughly, the idea is to introduce a new network-wide software layer that exposes one or more logical forwarding elements

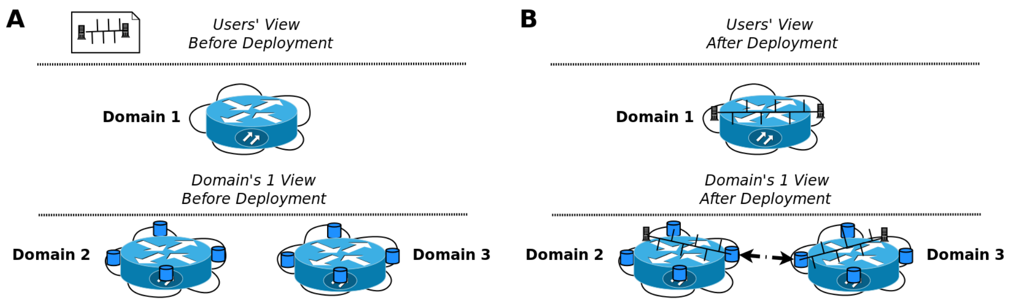

Figure 3.5: Single Router (Node) View of the Network of a Domain.

This darker blue cylinders (endpoint or ports of our single router) are the key elements that need to be defined and integrated at a later design stage in order to guarantee that users deploying services in the federated cloud get the network features that are truly relevant for their applications' proper performance. At an architectural level, the endpoints will include two major interfaces:

- External Domain (client) Interface: it exposes endpoint configuration or negotiation capabilities as a service for external users. For example, a client domain may decide to connect two neighbouring domains by using any available endpoint in that inter-domain junction.
- Internal Domain (provider) Interface: the domain receives the external (client) domain requests and maps it to a series of internal operations and actuations on a series of devices that bring the required elements up to work. For instance, an L3 VPN with some Quality of Service (QoS) guarantees is to be established with a remote endpoint indicated in the request.

While the specifics of the internal domain interface are operator-dependent, the external interface should be built following a bottom up approach. In other words, getting data from the available technologies implemented at these junctions (and across operators) so as to build a common set of abstract operations and data types to be used ubiquitously, granting that a request can be placed no matter the actual technologies and policies implemented by final (the one that does no delegation at all and owns the infrastructure) domain.

3.3.2 Goal Translation and Data Model Refinement Synchronisation

The level of specification and the details about the data model vary according to the level of our architecture. The information and the data model vision of a user requesting a service are different from those explored by the infrastructure provider. A high level description is refined and formatted with information coming from management framework. For exemplifying the expression of user's requests and the goal translation considering our data model, we selected a scenario were one is requesting a traditional IaaS, informing some elasticity configuration as well as the network requirements. At a very high level the goals expressed by the user could look something like the following:

- 10 VMs and connect them to my two enterprise sites, Madrid and London. VMs: 2 vCPU, 4GB RAM, 120GB HDD, Linux OS, Internet connection, located in Germany.
- I want to be able to scale up to 50 VMs (i.e. the minimum I want is 10, and I can go to a maximum of 50 - SLA).

- Connection between data centre and enterprise sites: maximum end to end delay of 100 ms. "I don't want my connection to pass through Belgium".

Using parts of our possible embodiment for the data model (see Section 3.2.1), this request would look something like presented in Figure 3.6 (shortened for the sake of brevity). This does not mean that this is the final expression of the data model, just a very illustrative example.

This model is in line with the high level description in Sections 3.1.1 - 3.2 with regards to how the data model should look like. Also, the presence of "access points" here can be understood by making the analogy with the ports in the single router abstraction we mentioned above. A VM in an external data centre (Germany) is linked to one of the ports in our facilities "single router" (the access link).

We can also clearly observe how the data model reflects the goals at this level of the architecture. However, in the refinement process, the management functions explained in Section 5 also need to be refined and re-expressed accordingly to the increased level of detail and the dynamic references being resolved as such requests get mapped into the actual infrastructure. At the end of the day, when a VM gets to an IaaS cloud provider, the provider itself needs to trace certain goals (e.g. minimize data centre energy consumption by keeping physical machine utilization up to 80 % and no less than 65 %, then switch equipment off. Of course, these goals imply different knowledge (e.g. specific placement algorithms, data centre topology information, etc.) and details that need to be mapped into the data model.

```
<?xml version="1.0" encoding="UTF-8"?>
        <description xmlns="http://www.vxdlforum.org/vxdl" xmlns:xsi="http://www.w3.org/2001/
            XMLSchema-instance" xsi:schemaLocation="http://www.vxdlforum.org/vxdl VXDL.xsd ">
                <virtualInfrastructure id="Goal translation. Example" owner="SAIL project">
                        <vArray id="10VMs" cardinality="10">
                                <vNode id="VM">
                                        <location>germany.eu</location>
                                        <cpu>
                                                <cores>
                                                        <simple>2</simple>
                                                </cores>
                                                <frequency>
                                                        <interval>
                                                                <min>2.0</min>
                                                        </interval>
                                                        <unit>GHz</unit>
                                                </frequency>
                                        </cpu>
                                        <memory>
                                                <simple>4</simple>
                                                <unit>GB</unit>
                                        </memory>
                                        <storage>
                                                <interval>
                                                        <min>120</min>
                                                </interval>
                                                <unit>GB</unit>
                                        </storage>
                                </vNode>
                        </vArray>
                        <vAccessPoint id="Madrid">
                                <externalRegion>Internet</externalRegion>
                                <location>madrid.spain.eu</location>
                                <ipsec></ipsec>
                        </vAccessPoint>
                        <vLink id="VMs to Madrid">
                                <bandwidth>
                                <forward>
                                        <interval>
                                                <min>1</min>
                                                <max>10</max>
                                        </interval>
                                        <unit>Mbps</unit>
                                </forward>
                                <reverse>
                                        <interval>
                                                <min>1</min>
                                                <max>10</max>
                                        </interval>
                                        <unit>Mbps</unit>
                                </reverse>
                        </bandwidth>
                        <latency>
                                <interval>
                                        <max>100</max>
                                </interval>
                                <unit>ms</unit>
                        <latency>
                        <source>10VMs</source>
                        <destination>Madrid</destination>
                </vLink>
        </virtualInfrastructure>
</description>
```

Figure 3.6: VXDL example of goal description.

4 Control Interactions

In this section we consider control interactions across infrastructure services. The arrangement of control in the CloNe architecture is based on a number of principles, the need for delegation and the need for distributed coordination across administrative domains.

4.1 Principles of Infrastructure Control

Here we detail the requirements of the general model for development of the control plane.

- **Limited information disclosure**: each Infrastructure Service Provider will be unwilling to expose details of its service, physical topology or traffic matrices. Hence, the Infrastructure Service Provider should be able to undertake cross-domain resource provisioning without detailed knowledge of underlying administrative domains.
- **Rapid resource provisioning**: the user should be able to provision resources across multiple domains at time scales comparable to existing compute resource provisioning in single domain IaaS offerings.
- **To scale with the number of Infrastructure Service Providers**: it should be possible for the Infrastructure Service Provider to utilize services from many Infrastructure Service Providers without major impact on the performance of the service.
- **To scale with the number of user requests**: it is important to give guarantees or bounds for searching processes and sending of requests and messages.
- **Autonomy of an Infrastructure Service Provider**: the Infrastructure Service Provider should be able to accept or deny any request from an infrastructure service user. Local policies can be applied locally if so desired.
- **Filtering**: a domain may apply whatever filtering policy it chooses to limit external interaction with its own domain, including cross-domain control interaction (similarly to the existing BGP).

A key design goal is to make the control infrastructure independent of the heterogeneous underlying networking technologies by defining and specifying interfaces and communications middleware with the appropriate level of abstraction.

4.2 Control Interactions

Infrastructure control can be broken down into two broad aspects in the CloNe architecture: *delegation-based control* and *distributed coordination*. The first of these refers to requests invoked by the infrastructure service user that result in actions performed against the user's virtual resources. These cascade down through the delegation hierarchy of infrastructure services according to the management functions of the services. The second refers to interaction among administrators in the distributed control plane to share or negotiate configuration information. The aspects are shown in Figure 4.1. The two aspects are related in that requests in delegation-based control may result in actions that require distributed coordination. The counterparts in a distributed coordination are identified by information in the data model passed through the infrastructure service interfaces in the form of references.

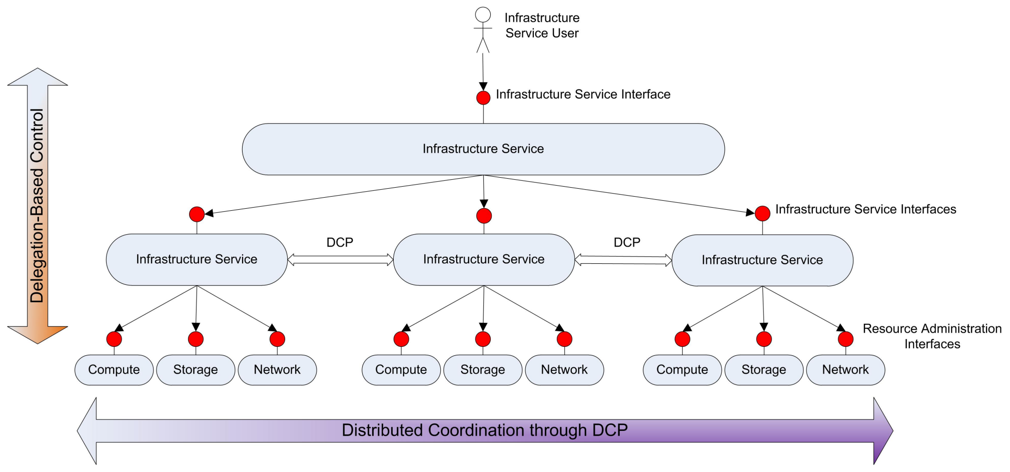

Figure 4.1: Two Aspects of Control Interaction Across Infrastructure Services

4.2.1 Delegation Based Control

Delegation is concerned with the devolution of management responsibility. It is a two-way relationship in which the infrastructure service user hands over management responsibility for an infrastructure (including its creation) to the infrastructure service provider, and conversely the infrastructure service provider reports on the status of that infrastructure to the infrastructure service user. The provider is free to chose how to implement the infrastructure and its management, including by further delegation to other infrastructure service providers, but is not compelled to inform the user of how this is done.

As management responsibility is passed down a hierarchy or across peers a delegation chain is established that is itself always hierarchical in structure, with control flowing down the hierarchy and reporting flowing up. The higher levels of the hierarchy have a more global view, but it is more abstract with less detail of implementation. At the bottom of the hierarchy are the infrastructure services with their own administrative domains that directly control the resources used to implement the service. These have complete knowledge of the resources under their control, but do not have a global view of the infrastructure.

Delegation based control is exercised through the infrastructure service interface. The act of delegation forms a trust relationship that is the basis of devolved responsibility, so a key feature of the infrastructure service interface is the capability for mutual authentication between user and provider.

4.2.2 Distributed Coordination

Distributed coordination is concerned with operations that involve multiple agents. In particular we consider operations that require coordination across multiple administrative domains at the bottom of a delegation hierarchy.

One obvious way to achieve distributed coordination is to involve a common control point to coordinate the actions; we observe that two administrative domains will always have a common infrastructure service user above them in the delegation hierarchy. However, if the lower infrastructure services are unwilling to communicate implementation details up the delegation chain, they may not be able to involve that common control point. More generally, once management responsibility has been delegated, it is no longer the responsibility of the delegating party to be involved in the management. So the delegation hierarchy can not be exploited to achieve distributed coordination.

In general we can assume that administrative domains will have to share necessary information to establish connections with their immediate neighbours, and so there will be some sideways sharing of information. We can assume that distributed coordination will follow a peer-wise communication pattern, using limited information sharing, outside the scope of the delegation hierarchy. This form of distributed coordination will occur through the distributed control plane.

In order to interact across the distributed control plane two peer infrastructure services will need to prove that they have the delegated authority to participate in management of a given infrastructure or resource, and they will have to identify themselves as the neighbour with which information can be shared. The first of these is inherited through the delegation hierarchy, the second is a peer-wise agreement between administrative domains.

4.2.3 Reference Resolution

An infrastructure service user may decompose its users' infrastructure models into multiple parts that will be delegated to different providers. These models may contain related information, such as shared configuration parameters or connected resources. Some shared information that is available at the point of delegation can simply be included in multiple models, but in some cases one provider will need information the is not known until another provider has further elaborated its part of the infrastructure.

This situation is dealt with by adding references to one model at the point of delegation that identify information held by the other providers. Resolving these references after delegation allows the information to be obtained from the target providers.

These references may identify the wrong provider or information in the event that the target provider transforms its model and repeats the delegation. In this case a new provider further down the delegation chain is the real source of the required information. This suggests a chained resolution process in which the target identified by a reference can redirect resolution to information held by other providers.

Care has to be taken with this process as it may inadvertently reveal information that providers would prefer to keep hidden. If a recursive approach is taken in which the delegating provider acts as an intermediary, the response may need to be obscured so that it can not be interpreted by the intermediary. If an iterative approach is taken, in which a delegating provider supplies a new reference identifying the delegatee, the provider will reveal the delegation hierarchy. If the resolution process involves an independent resolution mechanism, such as a publish subscribe service, these concerns may apply to the service itself.

In general the reference resolution process will need to follow indirection that results from delegation, but providers should be able to implement their own policies regarding information hiding. A provider may wish to ensure that certain information is only exposed to selected peers.

4.3 Interface Design

The control interactions among interfaces of the CloNe architecture provide for a set of functions (outlined in Section 2.5 and described in more detail in Section 5) that are responsible for the allocation and control of distributed computing and networking resources both within and across administrative domains. The form of these interfaces and the style of interaction across them should support the principles of infrastructure control described above over resources that are distributed and may be under the control of different administrative entities.

4.3.1 Delegation Interaction

The system design supports both peering and hierarchical arrangements of infrastructure service providers described in Section 2.3. In a peering arrangement each infrastructure service provider should establish resource sharing agreements with adjacent providers (not between domains that are more than one hop away). A distributed service is created by interaction across these peer-wise relationships.

In a hierarchical arrangement one infrastructure service provider composes a distributed infrastructure service based on resources provided by multiple infrastructure service providers. The one provider acts as a user of these providers and interacts with each independently. This infrastructure service provider does not need to own any of the infrastructure used to implement the service.

Even though communications providers prefer to offer services that reflect the current loosely coupled Internet structure (which speaks in favour of a peering model), the nature of the control interactions is likely to be influenced as much by business relationships as by technology. The hierarchical model allows for the creation of a new business entity which can compose services and choose different sources of infrastructure according to different metrics (e.g. price, reliability, security). Moreover, any existing infrastructure service provider (e.g., network provider, cloud providers) can implement that functionality.

Delegation through the infrastructure service interface is a synchronous act in which the user passes responsibility for management to the provider and typically later retracts that responsibility. Invocation of management actions in response to adopting responsibility may be asynchronous. Delivery of status reporting information to the user may be synchronous in response to query or asynchronous in the form of notifications.

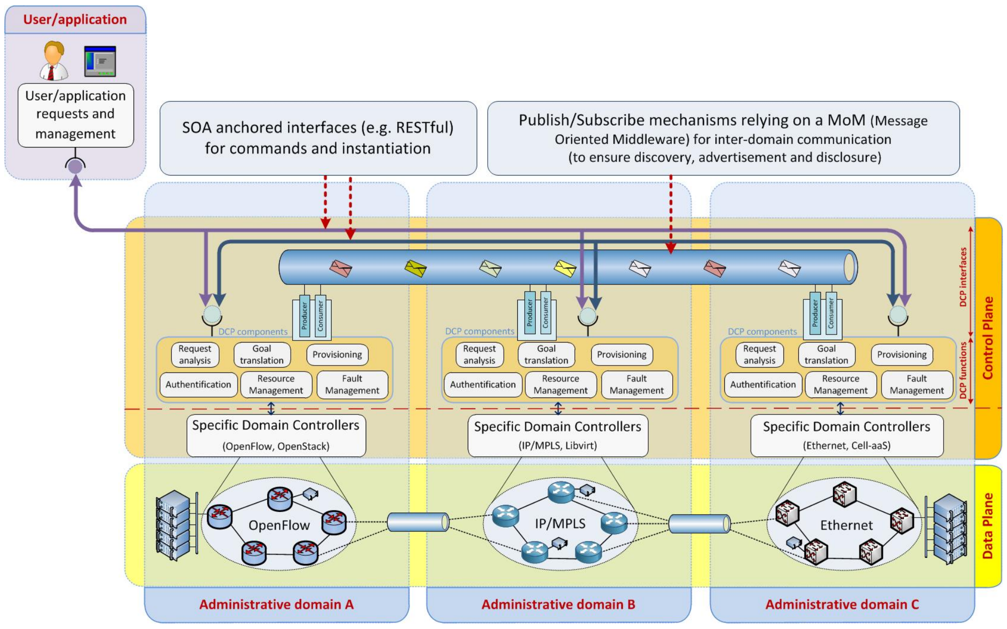
User/application
User/application requests and management
SOA anchored interfaces (e.g. RESTful) for commands and instantiation
Publish/Subscribe mechanisms relying on a MoM (Message Oriented Middleware) for inter-domain communication (to ensure discovery, advertisement and disclosure)
Producer
Consumer
DCP components
Request analysis
Goal translation
Provisioning
Authentification
Resource Management
Fault Management
Specific Domain Controllers (OpenFlow, OpenStack)
Specific Domain Controllers (IP/MPLS, Libvirt)
Specific Domain Controllers (Ethernet, Cell-aaS)
DCP interfaces
DCP functions
Control Plane
Data Plane
OpenFlow
IP/MPLS
Ethernet
Administrative domain A
Administrative domain B
Administrative domain C

Figure 4.2: Control interactions across domains

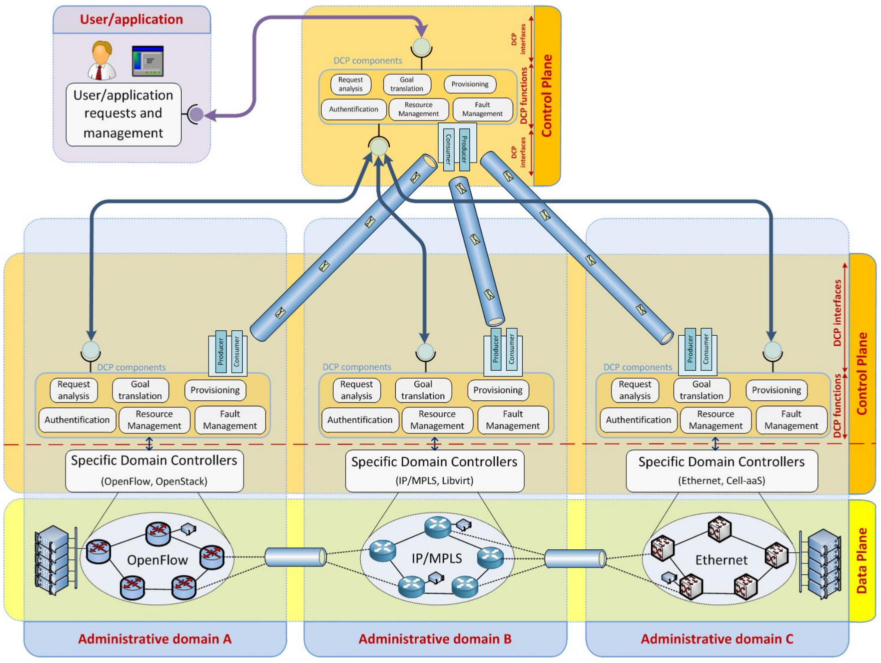

Figure 4.3: Hierarchical control interactions

4.3.2 Distributed Coordination

In addition to control interactions at the level of the cross-domain infrastructure service layer, there may be interaction at the resource layer to establish and manage links between resources in different administrative domains. These represent fulfilment of interconnection agreed at the higher layer. Both the forms of coordination are achieved through the DCP to cooperatively use of resources from several domains in response to requests received through the infrastructure service interface. The scope of this management depends on the established service level agreements between the domains since they determine the information shared and made available to the common control plane.

A key design goal is to make the control infrastructure independent of the heterogeneous underlying networking technologies by defining and specifying interfaces and communications middleware with the appropriate level of abstraction. A possible solution based on the current state of the art is to build the inter-domain interfaces as depicted in Figure 4.2 by relying on:

- Service Oriented Architecture (SOA) anchored interfaces, e.g. Representational State Transfer (REST) interfaces, for commands and instantiation in the common and distributed control part
- publish / subscribe mechanisms relying on a Message Oriented Middleware (MOM) for inter-domain communications

The SOA anchored interfaces are similar to the infrastructure service interface defined in Section 2.4.3 in terms of command and instantiation actions. The publish-subscribe mechanisms enable topology discovery across and within domains with information disclosed according to security and service agreements between domains. During this operation, information is hidden or disclosed securely and selectively depending on established Service Level Agreements (SLAs) between providers. The publish-subscribe system is typically designed to also control information disclosure

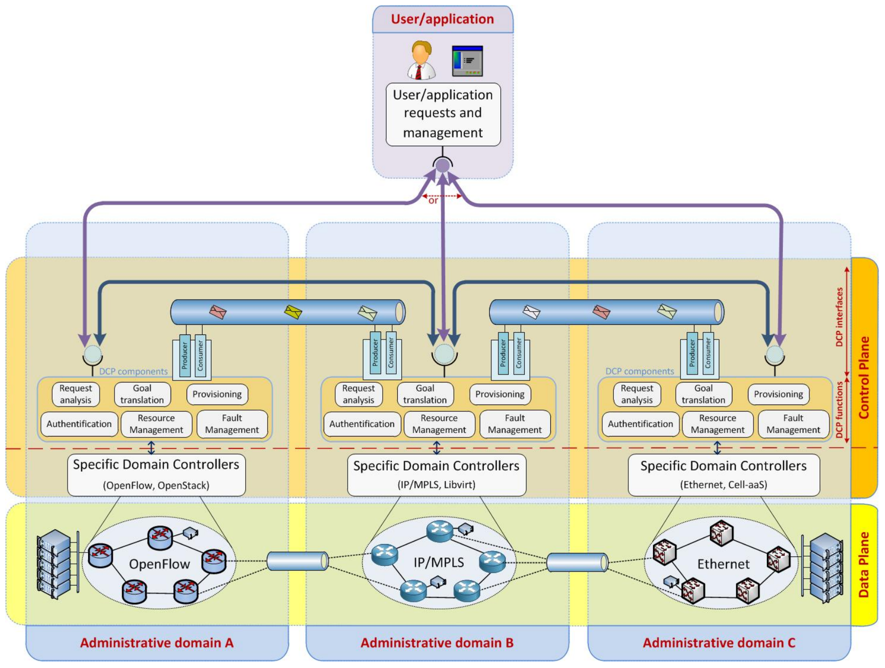

Figure 4.4: Peer control interactions

and advertisements and embeds a discovery and notification framework to facilitate inter-domain cooperation, interactions and information exchange, including reference resolution.

Interaction through the DCP is to achieve cooperation between two independent agents. Using a MOM decouples the interaction in time and space and provides the agents with the freedom to operate independently and implement their functions as they chose.

Both hierarchical and peering approaches can be adopted to design the DCP for inter-domain coordination and management. These two possible configurations for the DCP and its interactions with the domain specific controllers are depicted in Figure 4.3 and Figure 4.4 for the hierarchical and peering solutions respectively.

In the hierarchical solution, a common controller acts as a root for the system and handles all communications between the different domains and is the unique service access point for the service interface. In the peering approach, the DCP relies on the peer to peer paradigms where all controllers are logical peers that cooperate to achieve joint and cooperative management across the domains. The user can access the service interface provided by any participating domain.

4.4 Interface Standardisation

The infrastructure service interface and the DCP interfaces and protocols are points of interoperability between providers. In some cases these may be relatively ad hoc or proprietary, such as integration interfaces agreed between neighbouring administrative domains, in others they need to be generally defined and are therefore candidates for standardisation. The infrastructure service interface used to delegate infrastructure management to providers is one such candidate.

The infrastructure service interface is based on web technologies like REST since full support is readily available from key working groups and forums such as Open Grid Forum (OGF) OCCI-WG and Storage Networking Industry Association (SNIA) CDMI. This interface needs to be compatible with the cloud computing technologies and communities and it has to use the right level of abstraction to simplify interactions with applications and users on the user API side. In parallel, the interface needs to be useful for any lower level or domain specific technology via the lower level API. Several significant IaaS implementations implement the OCCI interface or are currently developing the interface, including OpenNebula [7] and OpenStack [20]. Most cloud resource and service tool and manager implementations today are compatible compatible with OCCI.

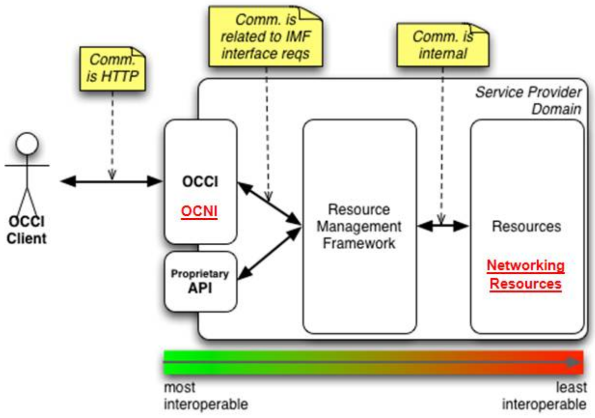

Figure 4.5: OCCI Protocol and API placement in a provider's architecture from [1]

OCCI is a boundary protocol and API that acts as a service front-end to a provider's internal management framework. Figure 4.5 shows OCCI's place in a provider's architecture. Service consumers can be both end-users and other system instances. OCCI is suitable for both cases. The key feature of OCCI is that it can be used as a management API for all kinds of resources while at the same time maintaining a high level of interoperability. For this reason it provides a possible starting point for CloNe to define an infrastructure service interface.

In the following we describe how OCCI can be extended to incorporate the CloNe capabilities. We call the extension Open Cloud Networking Interface (OCNI).

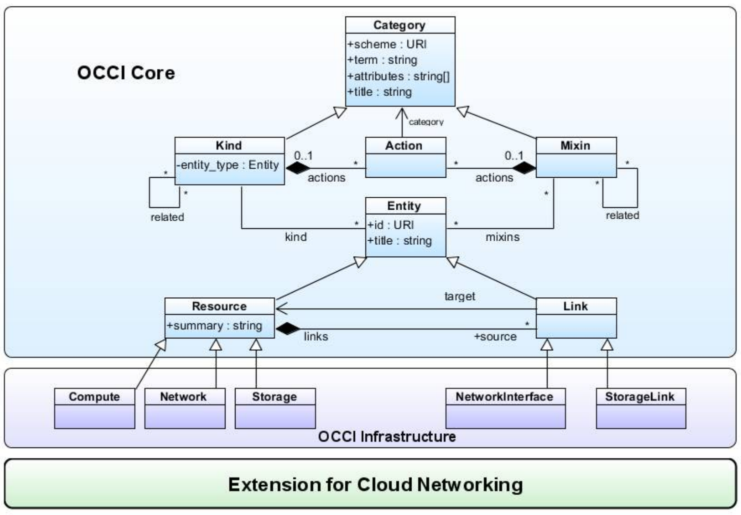

Figure 4.6: UML diagram of the OCCI Core and Infrastructure

OCCI describes cloud resources and services through an OCCI core specification supplemented by an OCCI infrastructure specification that views everything as a resource. The notion of resource in OCCI has a very broad meaning and describes a concrete resource that represents real world resources such as virtual machines, jobs in a job submission system, users, networks and even services (including applications, middleware services, network services, networking and connectivity services, etc.). The structure of the OCCI specification is depicted in Figure 4.6. The Mixin element allows the inclusion or integration of components from other domains or realms not originally included in the OCCI specification scope or coverage space. Despite this Mixin feature, it is essential to extend the framework to include cloud networking in the specification to move the clouds beyond cloud computing.

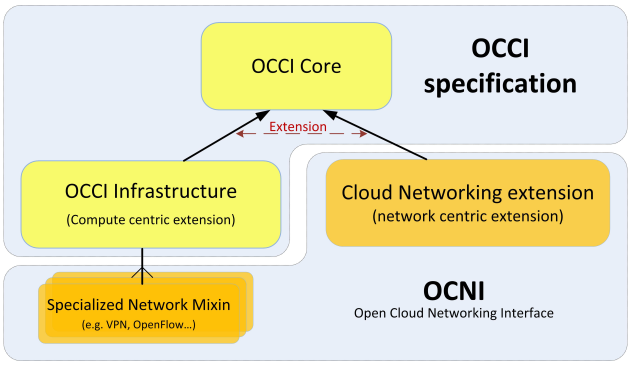
OCCI Core
OCCI specification
Extension
OCCI Infrastructure
(Compute centric extension)
Cloud Networking extension
(network centric extension)
Specialized Network Mixin
(e.g. VPN, OpenFlow...)
OCNI
Open Cloud Networking Interface

Figure 4.7: OCNI - An OCCI extension for cloud networking

The OCCI Core specification is enough to describe any cloud computing constituent or component in terms of attributes and characteristics but falls short of describing connectivity between these components. The OCNI extension depicted in Figure 4.7 is meant to fill this gap. OCNI is composed of two elements. The first and main one is a cloud networking centric extension to the OCCI core. This extension is in the same layer as the OCCI infrastructure extension which is compute centric. The second element consists of a number of specialized network Mixin that can be used in the OCCI Infrastructure extension as shown in Figure 4.7. Examples of such Mixins, most relevant to CloNe are for example VPN and OpenFlow network interface Mixins.

OCNI specifies an abstract data model taking into account expected and desired CloNe services to fill the existing gap in cloud computing by introducing cloud networking services. The ultimate goal is to merge networks and clouds to achieve convergence of cloud computing and operator networks and services. The details are depicted in Figure 4.8.

The OCNI extension adheres to and uses the OCCI original approach by adopting the same modelling framework but specialises the specification to the cloud networking domain by adding classes that relate to the CloNe networking resources and their relationships. These classes represent abstract concepts that can be used to encapsulate common networking concepts such as the FNS, including the data models presented in Section 3, or technology specific such as the virtual networking technologies presented later in Section 6. As such, model transformation during the delegation process, as described in Section 3, can be used to refine an abstract model to a more specific model while continuing to use the same interface for delegation.

The components of the data model in Figure 4.8 comprise those related to the OCCI specification and to the OCNI extension. These elements are described in terms of roles, functions and relationships for the reader's convenience and for broader comprehension of the data model. The OCCI Core and OCCI Infrastructure are repeated for reference as they are readily available and described in greater depth in [1] and [21]. The elements introduced though the OCNI extension use the same concepts as OCCI and are consequently compliant and compatible with the original

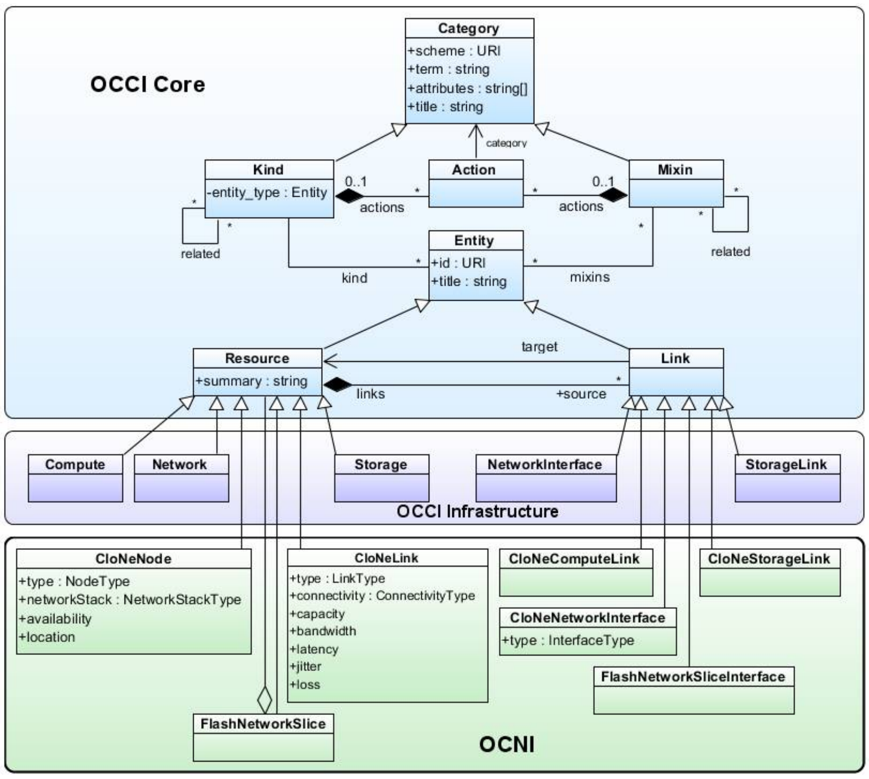

Figure 4.8: CloNe proposal for an OCNI extension of the OCCI framework

OCCI specification. This will enable natural integration OCNI and cloud networking concepts and facilitate the dynamic composition of FNSs to support the cloud community in setting on demand virtual and dedicated private and hybrid clouds. The components of the data model in Figure 4.8 are itemised below.

OCCI Core [1]

- Category: The Category type is the basis of the type identification mechanism used by the OCCI classification system.
- Kind: The kind type, together with the Mixin type, defines the classification system of the OCCI Core Model. The Kind type represents the type identification mechanism for all Entity types present in the model.
- Mixin: The Mixin type complements the Kind type in defining the OCCI Core Model type classification system. The Mixin type represents an extension mechanism, which allows new resource capabilities to be added to resource instances both at creation-time and/or run-time.
- Entity: The Entity type is an abstract type of the Resource type and the Link type.
- Action: The Action type is an abstract type. Each sub-type of Action defines an invocable operation applicable to an Entity sub-type instance or a collection thereof.
- Resource: The resource type inherits Entity and describes a concrete resource that can be inspired and manipulated. A resource is suitable to represent real world resources, e.g. virtual machines, networks, services, etc. through specialisation

- Link: An instance of the Link type defines a base association between two Resource instances. A Link instance indicates that one Resource instance is connected to another.

OCCI infrastructure [21]
Resource:

- Compute: Information processing resources.
- Network: Interconnection resource and represents a L2 networking resource. This is complimented by the IPNetwork Mixin.
- Storage: Information recording resources.

Link:

- NetworkInterface: connects a Compute instance to a Network instance. This complimented by an IPNetworkInterface Mixin.
- StorageLink: connects a Compute instance to a Storage instance.

OCNI
Resource:

- FlashNetworkSlice: a resource that provides a network service.
- CloNeNode: a networking resource of the Flash Network Slice.
- CloNeLink: a network link of the Flash Network Slice.

Link:

- FlashNetworkSliceInterface: connects a FlashNetworkSlice instance to a Resource instance.
- CloNeNetworkInterface: connects a CloNeNode instance to a CloNeLink instance.
- CloNeComputeLink: connects a CloNeNode instance to a Compute instance.
- CloNeStorageLink: connects a CloNeNode instance to a Storage instance.

4.5 Conclusion and further steps

We have described delegation through the infrastructure service interface and distributed coordination through the DCP interfaces and protocols. The infrastructure service interface is a clear point of interoperability between infrastructure service users and providers that warrants standardisation and we have described a possible approach to structure an appropriate standard. The interface itself and the data model passed over the interface can be separated to some extent by using an abstract representation in the interface, but a tighter alignment and agreement on data models is far more beneficial than the interface alone. Standardisation of this interface through data model and interface specification is currently being proposed by CloNe to European Telecommunications Standards Institute (ETSI).

The DCP includes a variety of interfaces and protocols as it encompasses technology specific interaction between administrative domains (such as BGP) and higher level management coordination between infrastructure services (such as resource life cycle coordination through management functions), some of which are public standards, some are proprietary. We have identified MOM and REST as suitable generic technologies to be supported in the DCP. As the management functions develop we will continue to re-examine the DCP to identify any future areas for standardisation.

5 Cloud Network Management

The CloNe management architecture is based on three management functions - goal translation, fault management and resource management. These management functions are in general focused on challenges concerning scalability, efficiency, manageability, adaptability and reliability, in order to cope with e.g. increasing network complexity, equipment heterogeneity, and fast reconfiguration in cloud networking. We will here give an overview of the initial management architecture. In Section 5.1 we present the management concepts. Section 5.2 presents the management architecture, the distributed management functions and relevant interfaces. In the Sections 5.2.5 and 5.2.6, we present general management processes and collaboration workflows between the management functions, followed by an overview of the initial approaches addressing cloud network management concepts in Section 5.3.

5.1 Management Concepts

As efficient and flexible usage of resources are some of the most important driving forces for the adoption of cloud networking solutions, management solutions should be designed to be highly efficient, scalable, adaptive and autonomous. The cloud network management should provide functionality for efficient and effective management of computational, storage and network resources, facilitate management of legacy networks, and utilise legacy management capabilities where needed. In order to be practically deployed, reliability and controllability must be ensured to a very high degree in all cloud networking solutions. Flexibility, reliability, and resilience can be achieved through decentralised, collaborative management solutions that to a high degree autonomously adapt to the dynamic conditions in the cloud network.

For the overall management of FNSs, we identify and present three important management concepts: goal translation, fault management, and resource management, which cover the most critical aspects of cloud network management, such as transformation of high-level objectives to low-level resource configuration objectives, security goals (further described in Section 7), configuration, monitoring, flexible resource allocation, and efficient optimisation.

Goal translation (GT) is needed for expressing high-level objectives in terms of low-level resource parameters for the configuration of FNSs, and facilitates dynamic optimisation and reconfiguration of the service infrastructure. Management via goals enables users with different backgrounds and agendas to request and manage services, without the need to deal with low-level aspects of composition and configuration of services. Competing service providers can delegate control of the service to its users without the need for disclosing the service infrastructure or business sensitive information. For robust self-management capabilities of cloud network management functions, uncertainty can be encoded into the goals taking into account the volatile service infrastructure environment.

Fault management (FM) is critical in providing resilience to faults and performance degradations, and to ensure reliability in cloud network services. Fast, accurate detection and reconfiguration is thus essential for maintaining service, connectivity and QoS. Fault management solutions need to be both reactive and proactive to network failures, for fast identification of the root cause and fault handling - preferably before the problem causes a noticeable degradation of the service or a violation to a high-level objective. Scalable and autonomous fault management solutions are necessary, to handle both growing network complexity and volatile network environments. Quick adaptation to changes in the network is crucial, and collaborative detection and localisation of faults and disturbances must be efficient to reduce communication overhead. It is therefore required that fault management solutions to a large extent operate in a decentralised manner.

The resource management (RM) concept is dynamic localisation and allocation of necessary resources in an efficient and timely manner. Scalable and efficient resource scheduling mechanisms enable fast location and prioritisation of available resources at a given time, ensuring short reaction times for FNS creation and adaptation in order to minimise disruptions in FNS 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 domains controlled by individual stakeholders exposing more or less limited capabilities and resource information, a homogeneous abstraction layer is required to abstract over potential heterogeneities for flexible FNS management. The unpredictability that follows from complex and dynamic network environments must be taken into account in order to avoid e.g., resource starvation. As asynchronous and concurrent creation of many FNSs prohibits the use of commonly used resource-blocking protocols, it is essential to design efficient algorithms that obtain a consistent snapshot of the resource situation while avoiding excessive temporary reservation of resources. To ensure that the resource allocation is optimised at all times, efficient live migration mechanisms operating seamlessly from the application and user are also necessary.

The combination of goal translation, autonomous fault management, and efficient resource management, has the potential to provide the foundation for a highly efficient management plane, in which resources are easily configured, managed, and monitored for malfunction.

5.2 Management Architecture

Conceptually, the management functions can be instantiated in any virtual instance or level of an infrastructure service provider, addressing management in both single- and cross-domain cases, given that the infrastructure service provider implements it according to its specific management purposes and with respect to the available infrastructure. Figure 5.1 provides an overview of the management functions described below (without specifying single- or cross-domain).

Management algorithms applied in a single-domain can not in all cases be directly used for cross-domain management purposes. For example, resource management and fault management require different types of information for management of resources and faults in the cross-domain case, compared to the single-domain. Moreover, the operation of the management functions may be restricted in cross-domain services, depending on the nature of the information the participating (single-domain) infrastructure providers are willing to expose, as reflected in the current commercial agreements between the domain owners. Limitations may be technical (e.g. varying network virtualisation technologies), administrative (e.g. operator limits knowledge it is willing to share with other domains), or legal (e.g. country-specific encryption limitations). A homogeneous abstraction layer, the Distributed Knowledge Plane (DKP) (Figure 5.1), is therefore required to abstract over potential heterogeneities for flexible FNS management.

The DKP is a generic term that represents the concept of distributed information retrieval and information maintenance of distributed resources, and allows for information transparency over the heterogeneous equipment on which the management functions operate. The DKP can relate to databases or functions for retrieving information about resources (e.g., for resource management), or relate to the common form in which heterogeneous information should be presented (e.g. for fault management collecting information from similar systems or legacy networks).

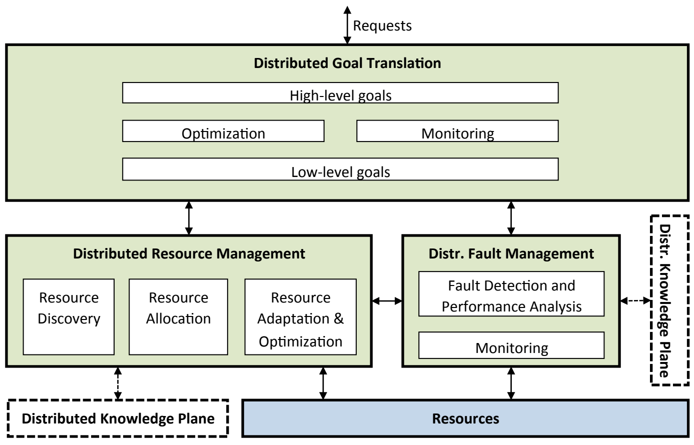

Figure 5.1: Conceptual overview of the management functions and the functional blocks.

5.2.1 Goal Translation Function

The GT function has three responsibilities. The first is to decide whether to accept a service request. The second is to translate the service request into lower level objectives with which the resources can be configured. The third is to monitor the fulfilment of high- and low-level objectives.

A *goal* is a set of constraints on one or several performance parameters of a resource, including both QoS and measurements of the status, properties, and connections of the resource. A goal is in this sense a high-level abstraction of a configuration of a resource. Note that goals need not only to be based on user requests, but can also be objectives received from other infrastructure providers. A performance parameter is defined in terms the parameters of a resource exposed by the RM function. Constraints on a performance parameter are called high-level objectives, whereas constraints on the set of parameters constituting a performance parameter are called low-level objectives. *Goal translation* is the selection of a set of low-level objectives for resource configuration, aimed at delivering a service performing optimally in accordance with a high-level objective. Note that a goal also can contain constraints on the exposed resource parameters directly, as e.g. in the case of an IaaS request, where a request could be in the form of a desired configuration of a VM.

The GT function correspondingly consists of four functional blocks: *high-level objective*, *optimisation*, *monitoring*, and *low-level objectives*. The high-level block is responsible for receiving and processing high-level objectives from the infrastructure service interface. The optimisation block is responsible for finding possible solutions offered by the resource management into low-level objectives that matches the high-level objectives. The monitoring block ensures fulfilment of the low-level objectives. The low-level block constitutes the translated objectives disseminated to the resource management. The GT function blocks require input from external sources according to:

- High-level objective block: a high-level objective describing the requirements on the requested service in terms of constraints on performance parameters.
- Monitoring block: performance parameter monitoring data from fault management for monitoring the fulfilment of the low-level objectives.
- Optimisation block: from RM a set of possible provisioning solutions for the low-level objectives. This is used for the selection of resources to be used in the delivered service.

The output from the GT function is provided by the low-level block and the monitoring block:

- Low-level block: low-level objectives used both as configuration instructions and as requests for solutions in the RM function, and identification of the selected provisioning solution.
- Monitoring block: goal fulfilment information based on the existence of a solution for the high-level objective (to be sent to relevant recipients such as infrastructure service users, management functions etc).

Goal translation for security purposes is described in further detail in Section 7.2. It is based on the same structure as the overall goal translation as described above, and focuses on generating the security specific configurations of underlying resources for facilitating secure resource management. It accepts a security control goal (also termed as a security objective) from the owner/issuer of the goal at a higher level in the control plane hierarchy, and translates it into sub-goals (which are further propagated to the lower levels in the control hierarchy), or parameters which need to be further constrained with respect to specific resources.

5.2.2 Fault Management Function

The responsibility of the FM function is to monitor the behaviour of resources and report disturbances to goal translation, resource management, security mechanisms, infrastructure service users, or infrastructure service providers in collaboration. The FM function provides measurements, performance models, as well as disturbance information, such as detected faults, anomalies, and changes in the observed behaviour of monitored resources. In the single domain, the FM function performs measurements of specific resources through the resource administration interfaces (Section 2.4). In the case of infrastructure service providers that span over several domains, a higher-level FM function can be instantiated but with a different implementation that addresses cross-domain management (Section 5.2.6). The FM function offers localisation and root cause analysis of detected disturbances caused by configuration changes, faults and anomalies, within and across virtual layers. Preventive actions and compensation for faults are done in collaboration with the RM function.

The FM function contains a Fault Detection and Performance Analysis (FDPA) block implementing necessary algorithms, and a monitoring block performing measurements on certain resources or groups of resources on request by the FDPA. The FDPA block is responsible for analysing and modelling observed resource behaviour, and can also monitor certain parameters on request from other management functions, infrastructure service users or providers in collaboration. Distributed information exchange between existing fault monitoring systems is facilitated through the DKP, addressing heterogeneity in information exposure between infrastructure providers (Figure 5.1). The DKP can here be regarded as a high-level abstraction of information, facilitating FM information exchange between providers that already have similar fault monitoring systems.

Upon a new service request, the FDPA block provides the RM and GT functions with monitoring and fault information for single or multiple resources. The FM function coordinates the monitoring of resources using distributed algorithms capable of adapting algorithm performance and behaviour to varying conditions in the cloud network or in a certain service. Fault monitoring algorithms are coordinated within the FM function to efficiently operate both within services and across service overlays. Based on alarm filtering and alarm correlation over shared resources, the FM localises detected faults and changes to a service or a resource in the cloud network. Probabilistic modelling and anomaly detection enables early detection of malfunctioning equipment and resources, which allows for taking preventive fault handling actions in collaboration with other management functions. Localised faults are reported to the resource management to trigger adaptation of relevant resources in order to fulfil a service and its high-level objectives. The FDPA block needs as input:

- Topology information - for organising and triggering FM algorithms of certain resources.

- Type of fault management algorithms - to specify the degree of monitoring and fault handling, and configuration inputs for selected fault management algorithms.
- Measurement responses from monitored resources,.

As output, the FDPA provides the following:

- Measurements and probabilistic models of observed resource behaviour and performance.
- Reports of disturbances, changes and degradations.
- Triggering of re-optimisation in the RM function for fault handling purposes.

5.2.3 Resource Management Function

The RM function manages provisioning, configuration and availability of storage, network and computation resources. It manages and keeps track of the properties of existing resources within its management scope. In single domains, resources are controlled through the resource administration interfaces (Section 2.4). Resource management in cross-domain cases requires a different implementation than management in single domains (Section 5.2.6). The RM function provides identification of a number of possible solutions with respect to a low-level objective. This is an iterative process, responding to a sequence of requests, each with its parameters, based on feedback from available resources, status reports from the FM function of monitored resources available for allocation, as well as goal fulfilment requests from the GT function. A solution to a low-level objective could include different alternatives (e.g. different price or performance). Further, the RM function provides clustering of multiple resources of multiple types (such that they are together allocated and de-allocated to a service request), resource assignment and de-allocation, optimisation of resource allocation (due to varying conditions in the the cloud), as well as balancing of traffic and workload among resources in order to fulfil a low-level objective.

The RM function consists of three functional blocks; resource discovery (RD), resource allocation (RA) and resource adaptation-optimisation (RAO). The RD block provides information about the existing physical and virtual topologies and characteristics of the resources. The RA block allocates and configures virtual resources related to a specific service request, taking the physical location of the resources into account. Given a request based on a number of possible criteria from both cloud and network (e.g. CPU, RAM and HDD for the cloud; latency and bandwidth for the network), the RA retrieves information from the RD to compute possible solutions that can be used as input to the GT function. Finally, the RAO re-optimises and adapts the use of resources either on a periodic basis or triggered by the fault management, to account for varying conditions in the cloud or network. The RM function needs as input:

- RD block: available resources, equipment and their properties, such as ID, entity information, handicap etc.
- RA block: low-level objectives for configuration of equipment; available resources; topology; fault and disturbance information; and performance measurements.
- RAO block: disturbance and degradation information about the equipment, information about resource usage and performance measurements.

The RM function provides as output:

- RD block: provides resource availability on request.

- RA block: possible solutions for optimised provisioning for goal translation, specification of resources to monitor and configurations to fault management, parameter settings for configuration of resources and equipment included in the accepted provisioning solution.
- RAO block: provides reconfiguration solutions to be allocated by the RA block, triggered by e.g. fault management.

5.2.4 Management Function Interfaces

Collaboration with other management functions and other parts of the architecture is enabled through the controller and collaboration interfaces introduced in Section 2.4.3. An overview of the interfaces (without specifying single- or cross-domain) is shown in Figure 5.2. These are the minimum interfaces currently identified for collaboration between management functions, and require further development. Typically, the kind of information available through these interfaces are objectives, topologies, resource properties, configurations, and performance measurements. Information exchange between management functions in different domains can be performed through the DCP (Section 4). Through the controller interface, management services and algorithms are created, configured and destroyed. The controller interface also provides information about the state of management through an API. The collaboration interface publishes continuous information to relevant recipients, such as updates about resource states, reports of potential faults etc. Relevant recipients are other management functions, entities or users subscribing to these services. The information is either published through a separate database function, or the management functions themselves manage subscribing recipients. Data can also be extracted on demand using API functions. Note that the communication between management functions has no direct relation to the DKP, as it is inherent to a management function only for keeping track of the status of heterogeneous equipment.

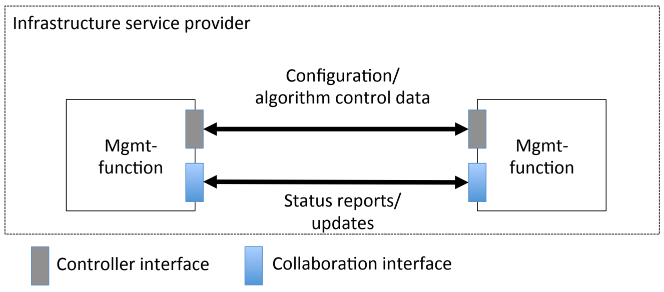

Figure 5.2: Example of management interfaces.

5.2.5 Management Processes

Figure 5.3 depicts a more detailed structure of the management architecture's key processes. The figure shows a conceptual view and does not specify whether processes are distributed or centralised. In a typical system, all information and processes are distributed across different domains (see e.g. Figure 2.3, 2.4). Process interactions and information exchanges are facilitated by appropriate algorithms that communicate via the collaboration and controller interfaces (Figure 5.2).

The GT function operates in three processes - the service request process; the goal monitoring process; and the re-optimisation process. In the service request process, infrastructure service user requests (given through the infrastructure service interface, Section 2.4) are processed into

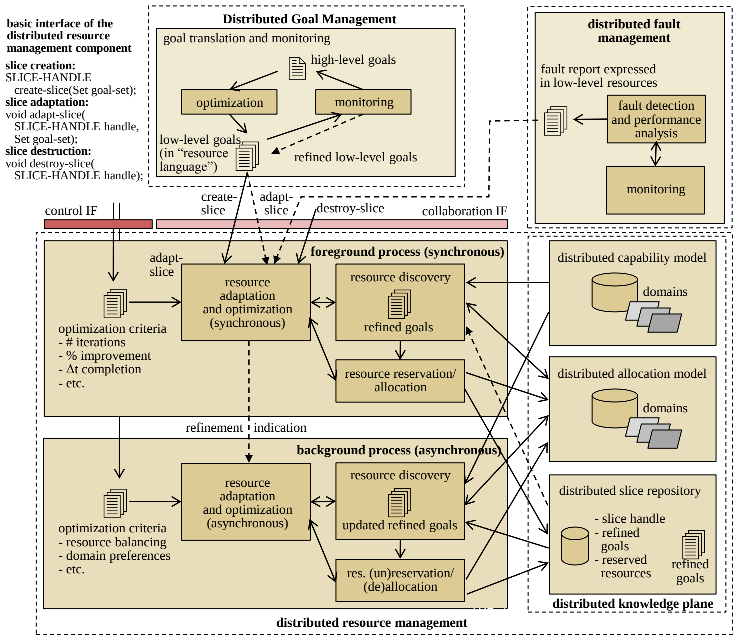
basic interface of the distributed resource management component
slice creation:
SLICE-HANDLE create-slice(Set goal-set);
slice adaptation:
void adapt-slice(SLICE-HANDLE handle, Set goal-set);
slice destruction:
void destroy-slice(SLICE-HANDLE handle);
Distributed Goal Management
goal translation and monitoring
high-level goals
optimization
monitoring
low-level goals (in "resource language")
refined low-level goals
distributed fault management
fault report expressed in low-level resources
fault detection and performance analysis
monitoring
control IF
create-slice
adapt-slice
destroy-slice
collaboration IF
adapt-slice
foreground process (synchronous)
resource adaptation and optimization (synchronous)
resource discovery
refined goals
resource reservation/ allocation
optimization criteria
- # iterations
- % improvement
- Δt completion
- etc.
refinement
indication
background process (asynchronous)
resource adaptation and optimization (asynchronous)
resource discovery
updated refined goals
res. (un)reservation/ (de)allocation
optimization criteria
- resource balancing
- domain preferences
- etc.
distributed capability model
domains
distributed allocation model
domains
distributed slice repository
- slice handle
- refined goals
- reserved resources
refined goals
distributed knowledge plane
distributed resource management

Figure 5.3: Overview of processes for management of flash network slices.

high-level objectives and used as input to the high-level objective block of the GT function. High-level objectives are translated into low-level objectives to which the RM supplies a number of possible solutions and the FM provides performance information on requested parameters. Through optimisation a set of feasible solutions is selected and offered to the requester. Upon acceptance, the resource is deployed and monitored according to the low-level objectives. The goal monitoring process ensures that low-level objectives are fulfilled based on the current conditions in the deployed FNS. If the FM or RM functions are unable to locally compensate for disturbances in the resource layer, or if the infrastructure service user makes a change in an already accepted goal, the monitoring process checks if any of the low-level objectives are violated, whereby the re-optimisation process of the high-level objective is triggered. If the high-level objective cannot be re-optimised and no alternative solution can be provided by the RM function, the monitoring block will notify the infrastructure service user about the failure to fulfil the high-level objective. The infrastructure service user can then choose to issue a new service request.

The FM operates in four processes - a monitoring process; a fault isolation process; a root cause analysis process; and a fault handling process. The distributed monitoring process is triggered as soon as the RM process has allocated and configured a set of resources for an FNS. The FM algorithms are configured for the set of resources specified by the RM together with algorithm configurations. The configurations can be part of the high-level and low-level objectives. The FM algorithms operate continuously on the monitored equipment in each FNS in an ongoing distributed and asynchronous process, running on the virtual resources. In this process, overall performance measurements are executed and modelled, used for autonomous adaptation of FM algorithmic behaviour [22], and for early detection of performance degradations. The fault isolation process is triggered when a failure or performance degradation is detected for a virtual resource within an FNS. The fault is localised in a distributed and collaborative manner within the FNS. The root cause analysis process is triggered when a fault has been localised. As the true root cause can originate from some other part of the cloud, the root cause analysis process correlates events from FNSs 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 (through the collaboration interface), actions to resolve the problem are taken.

The RM is split into a foreground and a background process, using information stored in the DKP. The purpose of the foreground process is to promptly find a valid resource allocation for an FNS defined by low-level objectives, handed to the distributed RM function. Discovery of candidate resources is performed for FNS creation, and the output of the resource selection process is handed to the resource reservation/allocation process, which in turn reserves the selected resources. In order for the foreground process to perform its tasks, optimisation criteria are needed to parametrise the resource selection process in terms of performance, such as a maximum number of iterations, a certain percentage of improvement over a first solution, a maximum time interval during which the synchronous foreground process must complete, or a combination of these parameters. The criteria can be modified through the control interface.

From the DKP, static information from the distributed capability model is queried, describing individual domains in terms of their capabilities in providing resources to the FNS to be created. The allocation model contains information about currently allocated resources in individual domains, but does not specify to which FNSs individual resources are allocated. The allocation model is updated by the resource reservation/allocation process when the resources for the FNS to be created have been reserved. In the case where an FNS is adapted via the adapt-slice interface call (Figure 5.3), the distributed slice repository provides additional information about the existing FNS, its handle, low-level objectives, and the resources reserved by that particular FNS. This information is taken into account in the adaptation process and is required to potentially de-allocate some of the resources while allocating some other resources during adaptation.

The purpose of the background process is to iterate through existing FNSs and attempt to optimise the initially allocated resources to achieve a more optimal overall resource allocation. The process operates asynchronously and performs autonomous FNS adaptation when the system load allows it. Whenever the foreground process is adapting the resources of an existing FNS, this is indicated to the background process via a refinement indication, to prevent the background process from operation on the same FNS concurrently. The background process works similar to the foreground process, but uses a different set of optimisation criteria (set via the control interface) to influence its behaviour. For instance, resource balancing and domain preferences may be used to control resource reallocation in a way that network and IT resource usage are balanced evenly across different domains. Refined low-level objectives stored in the distributed FNS repository are updated during an asynchronous resource selection process and resources are reallocated accordingly.

5.2.6 Information Exchange and Workflows

Service requests and constraints can be split and distributed between management functions in infrastructure services, where each part is self-managed based on delegated objectives, coordinating the virtual resources. The principal information exchange between the management functions within an infrastructure service provider is shown in Figure 5.4, with emphasis on the single-domain. Information exchange between domains is enabled through the available control interactions and the DCP (Section 4). Note that only the main architectural components are shown in Figure 5.4 and not the detailed processes shown in Figure 5.3 that map to it. From this view we present the workflows and information exchange between the management functions in three examples; one service request and two reconfiguration scenarios. Finally, we also show a conceptual example focusing on goal translation.

Initial request: (R)-(1)-(2)-(5)-(9)-(11)-(5)-(7)-(5)-(15)-(11): The GT processes the request (R,1,2) and sends it to the RA (5). The RA computes a set of solutions based on the available resources (9), possibly requesting input from (11), and hands the solutions to the GT (5). Perfor-

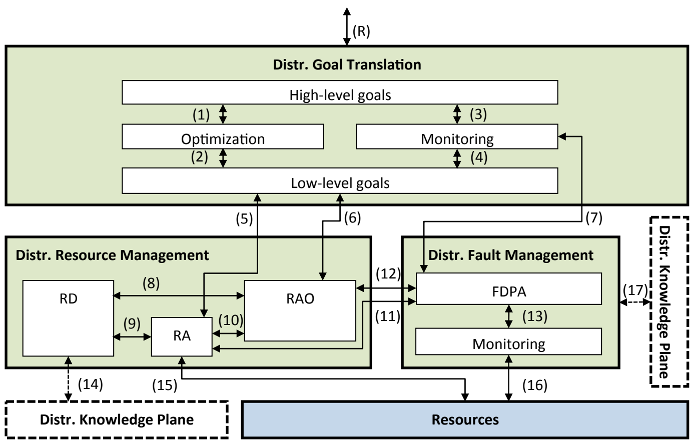

Figure 5.4: Principal communication between management functions.

mance measurements and models of observed behaviour are given by the FDPA (7), which will be an additional input for the decision mechanism in the GT. If the request is accepted the resources are allocated (5, 15) and the fault mechanisms are set up according to the allocated resources (11).

Reconfiguration for a detected disturbance: (12)-(8)-(10)-(15)-(11): A fault is detected, which will trigger a reconfiguration in the system (12). The current solution will be optimised with respect to available resources (8,10,15), followed by reconfiguration of the fault mechanisms (11). In case no solution is found, this is reported to the GT function. The monitoring block in the GT decides whether the disturbance is significant enough to violate the objective, generating a report to the user and possibly offering creation of a new service.

Change in high-level objectives: (R)-(1)-(2)-(6)-(8)-(12)-(6)-(10)-(15)-(11): The infrastructure service user makes a change in the high-level objective, which is translated to low-level objectives (R,1,2) and used as input to the RAO (6). Solutions are requested using input from (8) and (12), and are given as feedback to the low-level goals block (6) in the GT function. If the current solution fulfils the new high-level objective the solution continues to be in use. Otherwise, necessary equipment and fault management mechanisms are reconfigured in (10), (15) and (11).

In the cross-domain case, the principle of communication between the management functions is similar to that of single-domains, but with different types of information and algorithms. The information rather consists of high-level descriptions of available resources and their fault or availability status. The GT function is the focal point of communication - it gradually splits the objectives to relevant infrastructure service providers based on high-level information from the resource management and fault management, and disseminates it to the next lower level. Partial solutions obtained from the single-domain level (as described above) are aggregated and disseminated up to the topmost GT function in the hierarchy of infrastructure service providers. The acceptance of a certain solution is disseminated down the hierarchy of management functions and is implemented as described in the single-domain case above.

Goal translation example: Following the example goal expressed in Section 3.3.2, the high-level goal “Give me 10 VMs and connect them to my two enterprise sites in Madrid and London” is split into subgoals - here in terms of the request for 10 VMs, and as a request for the connection from Madrid to London given a set of constraints. Subgoals can be pure resource requests (i.e. low-level goals expressed as constraints on the parameters exposed by a resource API), or high-level goals expressed as constraints on parameters for performance characteristics of services or resources (see Sections 5.2.1, 5.3.1). A VM is a resource instantiated by the RM. The DCP receiving the goal needs to check with its subordinate infrastructure providers to see whether 10 VMs can be supplied (and scaled up to 50 VMs) according to the configuration instructions of a VM.

Given a necessary topology description from RM and performance analysis data from FM, the next step is to identify a candidate set C of FNS compositions X of resources that can provide the connectivity between Madrid and London, taking the constraints "maximum end to end delay of 100 ms" and "I dont want my connection to pass through Belgium" into account. For each composition X in C, the aim is to determine the set $W(X)$ of administrative domains that will be in control of the participating resources in X. To do this, each administrative domain D in $W(X)$ will receive a subgoal to fulfil.

Now, if X is a composition in C that makes one intermediate hop via Nantes on the way from Madrid to London, the combination $W(X)$ of administrative domains W and the links L between them could be written as $WMadrid - L1 - WNantes - L2 - WLondon$. Here, each element needs to receive its own objective (or subgoal) for the fulfilment of G.

To obtain a solution, optimisation is performed with respect to the elements in $W(X)$ and the extracted set of feasible solutions, respectively. For this, each administrative domain in $W(X)$ is required to report to the GT the capacity, such as available bandwidth, delay, drop and the cost of use with respect to the goal, based on information from RM and FM. The information can be probabilistic or deterministic. Optimisation based on the supplied information associates each candidate X with a feasibility flag, a cost and a probability of success (in case probabilistic information is available). The best feasible solution (or a set of solutions) is offered to the infrastructure service user, and is deployed if accepted. In runtime, the subgoals in the administrative domains in $W(X)$ are monitored to ensure that the high-level objective is fulfilled.

5.3 Initial Approaches

Until now, we have presented high-level management concepts of goal translation, fault management, and resource management. The following sections provide descriptions of initial approaches aimed at addressing these management concepts and the challenges of cloud network management.

5.3.1 Goal Translation and Monitoring of High-Level Objectives

Goal translation is concerned with the configuration of resources based on a high-level objective of a requested service. The high-level objectives consist of constraints on parameters expressing QoS, Quality of Experience (QoE), or otherwise measurable characteristics of the service. Sets of such constraints are called *goals*. The key issue is to find effective and sound translations of goals into constraints on the parameters and methods exposed by the service and resource APIs. In the hierarchical architecture, suggested for CloNe, the translation mechanism will propagate a goal through several layers. The successive translation steps should end in configuration constraints, which are passed to resource management for the configuration of cloud resources. Goal translation will be performed by a set of algorithms, collectively called the GT function. The GT function will be a part of the management API of an infrastructure service provider.

For goal translation, one of the main challenges concerns goal specification. On the one hand, the goal specification language needs to be rich enough for a manager to be able to express a desired behaviour of a service. On the other hand, the goal specification language needs to be restricted so that only goals that can be interpreted and implemented by the underlying cloud resources may be expressed. A complicating factor is that the service infrastructure may vary dynamically, not only in topology and physical equipment, but also in respect of QoS due to concurrent use of network nodes. In decentralised settings, there will thus be an inherent uncertainty about whether a goal can be fulfilled.

5.3.1.1 Approach

As described above, a goal is a set of constraints on parameters used for describing the behaviour of, or for configuring, a requested service. Depending on which resources are managed by a particular infrastructure service provider, exactly which parameters that can be used for formulating goals vary. If the parameters in a goal G (i.e., the parameters occurring in some constraint in G) are all such that the resource management function can configure a requested service according to those parameters, then the goal is said to be a *low-level goal* (or *low-level objective*). Correspondingly, we call those parameters *low-level parameters.* A requested service can also be described in terms of parameters that not necessarily can be used directly for configuring a resource, for example, bandwidth or cost of a service. Goals containing constraints on such parameters (high-level parameters) are called *high-level goals* (or *high-level objectives*). Goal translation aims at translating high-level goals into low-level goals and by that enabling the configuration of resources based on high-level goals. For the applicability of goal translation, it is important to restrict the parameters on which high-level goals may be expressed according to the present resource availability.

The goal translation function will offer to fulfil a goal, or in other words offer a service, with a probability. This probability reflects the inherent uncertainty of the cloud environment, where the resources are used and allocated competitively in a decentralised fashion. In a market based setting, we can assume that the same goal (or service request) may reach several infrastructure service providers, and that the requester may choose among several offered services based on the declared probability of fulfilling the goal. Such a decision may also be based on cost. A more detailed description is provided in Appendix A.1 that gives an account for our approach to goal translation and the restrictions on high-level goals.

5.3.2 High-Level Specification

Informing the desired goals (set of constraints, performance and configuration parameters) is an essential step in FNS provision. As the goals are interpreted and propagated among different levels of the management framework, a robust and efficient modelling language is highly required for carrying the substantial information among different actors.

The FNS model comprises some information that can be parametrised during the goals definition. Located in the highest level of specification, a user can define initial goals that represent his expectations in terms of QoS and QoE, such as:
(1)Reservation slot: the connectivity service is available during a given reservation slot. (2)Access points: the data centres interconnected by an FNS can have different network protocols and techniques. This information must be defined for each access point during the service requirement. (3)Latency and bandwidth: both performance parameters can be defined between specific access points or for the general FNS. (4)Location: motivated by several reasons (e.g., data dependency, governmental laws, user location) some FNS components must be provisioned around a specific location [23]. (5)Security: a user can configure the security goals and policies required for the FNS, (e.g., data encryption method, limits of data moving and location, network encryption). (6)Monitoring: the information on FNS internal resources (performance and usage) must be available for users during and after the execution slot. Configuring the metrics and rules for composing the monitoring policies is highly required. With this information a user can refine its FNS composition for adapting it to dynamic applications requirements. (7)Elasticity: an FNS can be dynamically adapted for a new configuration during the execution time. Usually, this configuration can vary based on the application behaviour (e.g., new workload, peak of usage, new users). A user should specify the rules for defining the elasticity of FNSs.

5.3.2.1 Approach

Our approach in the context of CloNe is that, a modelling language can be used for carrying FNS goals through different levels (e.g., from users to CloNe, among infrastructure providers, and between management framework modules). Consequently, a language for describing the FNS composition must be abstract enough and more adaptive than conventional resource-description languages and models. In addition, the model must carry substantial information related with the service provisioning and configuration, such as protocols definition, routing and forwarding tables, scheduling rules, and inbound/outbound rules (e.g., IPSec, NAT, load balancer).

For accomplishing the requirements of a high-level specification, we propose the integration of the VXDL language [8] [24] with the CloNe management framework. VXDL allows the description of a virtual infrastructure or a *compute-and-communication* resource graph, and it fulfils most the requirements of FNS goals specifications. Some extensions are required for specifying the elasticity aspects of FNSs, the monitoring expectations, and the required security configuration, and can be simply added to the VXDL data model. This language proposes a simple and high-level grammar to specify virtual infrastructures hiding hardware details, allowing for an efficient description of virtual infrastructures; more specifically, the identification and parametrisation of virtual resources and groups of resources (according to their functionalities), as well as the network topology (based on the link-organisation concept), using the same grammar. VXDL also introduces the internal virtual infrastructure timeline, which explores the elasticity of virtual infrastructures, enabling application providers to specify the exact intervals where virtual resources must be activated [25]. An important feature of VXDL is that it proposes cross-layer parameters for all components. For example, with the specification of *location* and *exclusivity*, users can directly transmit through different levels enabling an interaction between the user and the management framework.

5.3.3 Fault Detection in Virtualised Systems

As networks and connected resources are virtualised to a larger degree, determining the root causes of faults and disturbances at both network and service levels become increasingly complex. In cloud networking, this is specifically challenging as the network environment is continuously changing, with connecting and disconnecting equipment, and with dynamically configured FNSs. Detected faults, disturbances and performance degradations that appear for a certain service, may have a true root cause originating from virtual network resources in other parts of the cloud on which the service depends. Such faults could for example be caused by configuration errors, resource depletion or malfunctioning equipment. 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. Moreover, the process needs to be decentralised and autonomous, which introduces further challenges such as information dissemination between layers and participating network elements, timing, synchronisation, and correlation of detected events. Information exchange needed for successful fault management across different infrastructure service providers using similar fault detection systems is facilitated through the distributed knowledge plane (see Section 5.2.2). We here present an initial approach implemented in the FM function of the management architecture, for detection and localisation of faults in virtual and physical resources.

5.3.3.1 Approach

In the initial approach that we propose for detecting and localising the origin of faults in multi-tier overlay networks [26], each layer performs decentralised fault detection and localisation [27, 28], while keeping track of events in the network (for a limited time). These events include anomalies detected in QoS parameters based on both local measurements and aggregates, detected faults, and configuration changes. As a fault is detected in a higher virtual layer or FNS, events in the lower virtual level that are relevant given the time frame and location of the detected fault are correlated.

It is assumed that each virtual layer performs distributed fault detection and fault localisation [27], operating in the FDPA block of the FM function. Faults in a virtual layer A, running on top of level B, are localised down to a topology T_f^A, which could be e.g. a single node, link or a set of virtualised resources. The topology constitutes resources affected by the fault, and is identified through distributed collaborative fault localisation [27] between virtual or physical nodes.

The fact that a fault has been detected is disseminated to the underlying virtual layer along with the topology T_f^A and the timestamps $[t_w, t_f]$ (where $t_f > t_w$), corresponding to the time of the detected fault t_f and the time when everything was known to be functional t_w, respectively. This dissemination is performed locally on the virtualised node, as the underlying level knows how to relay this information further.

As the underlying virtual layer knows what the topology T_f^A corresponds to in terms of its own topology T_f^B, it searches the virtual resources for events that occurred between t_w and t_f. The basic assumption is that a fault in a certain topology in a higher virtual level should manifest itself as fault, anomaly, or configuration events inside the corresponding topology T_f^B in the lower level.

A set of correlated events E_f^B is found by searching all events originating or affecting topology T_f^B within the time frame t_w to t_f. This set constitutes a reported "root cause" at virtual level B, and is disseminated to the virtual level C below in the same manner as the original faults above. In the Appendix A.2 we show the scalability of the event correlation protocol.

By studying larger number of event sets within a layer we can potentially find an ordering (pseudo-causal graph) of the events. Over time, common event sets E_f^B can be clustered at virtual layer B, where each cluster forms a typical error expression, for the purpose of e.g. further analysis or faster determination of the root cause. Next step in our work is to further investigate how such analysis can be done in a decentralised manner.

5.3.4 Scalable Resource Allocation under Management Objectives

In the context resource management at the infrastructure service provider layer (see Section 2.1), a key problem is that of dynamically mapping the requests for virtual infrastructure to the available physical resources in the cloud. Typically, requirements of service users are modelled as a set of virtual machines and their associated storage and connectivity requirements. The responsibility of resource management is, based on a set of *management objectives* of the cloud, to (1) allocate the resources of the cloud to new service users and (2) to continually reconfigure the cloud in order to adapt to changes (see Section 5.2.3). Possible management objectives for a cloud environment include:

- To allocate resources fairly (or weighted fairly) among the service users. This may be suitable for best-effort clouds.
- To minimise the energy consumption of the cloud. This may be suitable for a 'green' cloud.

The key challenge in engineering the resource management solution for the infrastructure service provider is that of developing protocols for resource allocation and adaptation that are efficient, produce resource allocations with sufficient quality and are scalable to a cloud composed of some 100,000's physical machines providing services to a comparable number of users.

5.3.4.1 Approach

Our approach is based on the following two principles. First, as the resource allocation problems are often NP-hard, it is not possible to provide optimal solutions for environments of reasonable sizes. As a result, we focus on engineering efficient heuristic solutions that approximate well the optimal solution.

Second, the goals set out above for the resource management solution are difficult, if not impossible, to achieve using traditional, centralised management architectures. In centralised management systems, the computational complexity of management tasks resides almost entirely in management stations, as opposed to in elements of the managed system. Since the load on a management station and the time needed to execute most management tasks increases (at least) linearly with the system size, the overhead and the delay associated with such tasks can become prohibitively large when the system reaches a certain size. To address the outlined problem of scalability, we rely on the use of distributed and adaptive protocols for resource management. A key property of such protocols is that each element only maintains partial knowledge of the networked system and thus interacts only with a (small) subset of all elements in the system.

In our recent work for SAIL [29, 30], we focus on the problem of allocating CPU and memory resources to applications under varying management objectives. For this problem, we proposed efficient gossip-based heuristics that execute in an in-network management framework, similar to that followed in the FP7 4WARD project[2]. The performance evaluation of the protocols showed that they achieve the design goals outlined above. Specifically with respect to scalability, the evaluation results show that the performance of the protocols does not change with increasing system size, allowing for an efficient execution in a cloud consisting of more than 100,000 machines.

Our plan in SAIL is to extend this work in a number of ways. First, in addition to CPU and memory resources, we will consider network and storage resources. Second, we will expand the applicability of our solution to a wide range of application frameworks, such as web-applications, MapReduce, etc. Third, we will study the applicability of our management approach to the distributed infrastructure service provider layer (see Section 2.1).

5.3.5 Distributed Oblivious Load Balancing in Cloud Computing

The problem of effectively managing distributed computational resources has been extensively studied. Nowadays, it is receiving renewed attention due to the growing interest in large-scale data centres, in particular in the context of Cloud Computing.

To cope with high loads, service providers construct data centres with tens of thousands of servers [31]. Such data centres, which often provide a single service, may be located in different places around the world. This is the case, for example, with the Google search services - at each given point in time, there are several front-end servers active in various locations over the globe, and an incoming user search request is directed to one of them.

The actual choice of the specific server to process a given request has a critical impact on the overall performance of the service. A congested server not only introduces a service delay, as jobs are waiting their turns in the queue, but also creates a networking bottleneck around it. This network congestion further deteriorates the service, due to dropped packets. In scenarios, where the cloud provides both networking and computational resources, such bottlenecks are quite painful, as they negatively affect the performance of both types of resources.

The server to service-request assignment is a very difficult optimisation problem; there are multiple objectives and many parameters that should be considered. For example, the current load of available servers, or current network latency. These parameters can be reported, estimated or learned. Regardless of the exact optimisation 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.

5.3.5.1 Approach

The size of data centres, coupled with their distribution across the globe, call for fully distributed load balancing techniques. Considering the volatility of such systems, the state information collected for the optimisation problem must be frequently updated. We therefore would like to study the performance of oblivious distributed load balancing systems.

An oblivious system (also termed static system) is a system that ignores the current state, and does not use any dynamic input. Clearly, such oblivious load balancing solutions improves the overall performance of the services and the data centres, while obviating the collection of state and other information that are needed in legacy load balancing systems. Our approach complements the algorithm described in Section 5.3.4, in which some state information that is dynamically changed is taken into account. Despite this basic difference, both algorithms implement a distributed scheme, and seems promising.

We plan to deploy a novel approach for addressing oblivious load sharing, utilising a two-priority service discipline at the servers, and duplicating each job request into two copies, each with different priority. Both job requests are sent randomly to different sites. Once one of the replicas is completed, the other job request is stopped. Our expectations are that at least at some cases, where the high priority copy arrives at a highly loaded server, and the low priority copy ends up in a lightly loaded server, the low-priority job may be completed first. Clearly, in such cases, the performance of the service is improved. Since at each server high-priority jobs are always served before any low-priority job, the performance of our system is always at least as good as the basic random assignment technique and has the potential of offering considerably improved performance. Informally, one can think of the low-priority job scheme as an auxiliary mechanism that uses ”leftover” capacity to increase the overall response time of the system.

Our load-balancing scheme is oblivious, but it is not overhead free; we need a signalling mechanism for removing the redundant copies of completed jobs, and a (nonstandard) buffering mechanism. We have to take into account this overhead, when quantifying the benefits of our system.

5.3.6 Live Migration of Services in the Cloud

Cloud computing platforms allow hosting of multiple services on a globally shared resource pool, where resources are allocated to services on demand. Recent advances in the server virtualisation technologies radically improve the flexibility and versatility of resource provisioning. This is done through the ability to collocate several virtual machines on the same physical host, to dynamically change virtual machine resource allotments, and to migrate virtual machines across physical servers.

Migration can equalise resource utilisation across different data pools, or shut-down the under-utilised portions of the infrastructure to save operational costs related to power, by consolidating the same number of virtual machines on a smaller number of physical hosts [32]. Migration can also be an affective mean to control heat dissipation in passively-cooled data centres.

Live migration is also useful in several important management cases: Host evacuation due to maintenance, where the corresponding virtual machines can be migrated to other locations; or performance optimisation due to changes in the network conditions (e.g. avoiding a congested or broken link, or for improved QoS parameters, such as lower packet delay or higher bandwidth). Live migration of virtual machines is a useful management tool. However, without proper implementation, live migration overhead may cause the services to violate their SLAs.

It is clear from this analysis, that live migration is quite beneficial. The challenge is, however, to facilitate such migration in a manner that is fully transparent to the service. If this objective cannot be met, then service interruption during the migration should be minimised.

5.3.6.1 Approach

The migration process basically consists of transferring the memory image of the service from the source host to the destination host. In the off-line migration process, the source virtual machine is suspended, the entire memory image is copied to the destination physical host, and then the copied virtual machine is restarted on the destination host. With live migration, most of the migration process takes place while the source virtual machine continues to run, and the service is alive; the service is suspended for a short period of time before it is restarted on the destination host. Clearly, live migration has an advantage of maintaining the service availability, with only a short period of service downtime, which might be acceptable or recoverable at the service level.

One approach for live migration is the implementation of pre-copy process, in which memory pages are iteratively copied from the source to the destination server, without stopping the execution of the virtual machine [33]. The page copying process may take several iterations, during which dirty pages are continuously transferred. At one point, the server is stopped until all the pages are fully transferred to the destination, thus completing the copy phase. Subsequently, the virtual machine can be resumed at the destination host. Post-copy migration is also possible, in which the service is migrated before the memory pages are copied [34]. However, the migration of the memory pages consumes computation resources, and thus may degrade the service performance.

Furthermore, if live migration is being performed in-band (i.e., the same network bandwidth is being used by the migration process and by the service running in the virtual machine) then we expect even more severe service degradation, due to the fact that the migration process consumes some of the bandwidth used by users of the service.

There is a non-trivial trade-off between minimising the copy phase duration and maintaining an acceptable quality of service during the copy phase. We plan to investigate, model, and optimise the live migration process, in a quantitative study. We plan to start by evaluating the expected degradation in service level due to bandwidth restrictions. We will first address a simple model, in which the bandwidth used by the migration process is fixed throughout the entire pre-copy phase. Once we gain better understanding of this simple model, we plan to develop an optimal migration strategy for the general case, where the amount of bandwidth used by migration varies over time.

5.3.7 Predicting and Modeling Virtual Infrastructures

In most applications IT resource usage is a non-stationary quantity. Depending on the type of application, the generated workload can be a highly varying process that makes it difficult to find an acceptable trade-off between an expensive over-provisioning (for peak load) and a sub-performing resource allocation that does not mobilise unused resources. A dynamic bandwidth allocation approach in the context of network virtualisation can be a solution for this issue.

Using the elastic-video use-case as a proof of concept, we want to adaptively tune the provisioned bandwidth to the current application's workload.

Considering this scenario, we define the application workload as the number of simultaneous downloads. We select a flexible model that realistically describes the user's behaviour. An important outcome of this model is a theoretical Markovian description of the instantaneous generated workload. Thanks to this model, we expect to statistically quantify the occurrence, the amplitudes and the duration of all possible states of the workload, and more specifically its large deviations from the nominal phase. This mathematical model, along with a real-time monitoring of the system, will serve as the main input to adaptively configure FNSs to guarantee the QoS and QoE of users.

5.3.7.1 Approach

We propose three main approaches for achieving the challenges identified: an epidemical modelling for workload generation, an extension of the VXDL to enable the modelling of elastic aspects of an FNS, and the configuration and preparation of a testbed for validating the proposed approaches.

Epidemical modelling for workload generation: Information dissemination in a social system (Gossip or Buzz) can be viewed as an epidemic spreading in a population network. We studied relevant epidemic models and then drew analogy between epidemic outbreaks and user behaviour patterns in a video on demand system. Spreading of an epidemic in a population can be categorised into a few characteristics which is related to the infection under consideration. We elaborate on this model in the Appendix A.3 section.

Modelling elastic aspects of FNS with VXDL: VXDL is a high-level language for modelling and describing virtual infrastructures that defines attributes for coupling with virtual resources (e.g., links, routers, nodes, access points) and all complexity and diversity related with the definition of these entities [8]. We aim to participate on the VXDLforum (`http://www.vxdlforum.org`) for investigating the extension of VXDL for dynamic provisioning and elasticity support of FNS. An extension of VXDL for describing elasticity should comprise the definition of triggers and rules for each virtual resource and attribute. Triggers can define *when*, the moment in which an FNS must be reconfigured. A trigger should be defined in terms of capacity, time, load variation, and resource usage. When the trigger is activated a set of actions can be applied for reconfiguring the FNS resources. As VXDL already enables the specification and modelling of virtual infrastructures including the specification of their internal timelines, it is natural that this language can be extended for incorporating the elastic aspects of FNS. The introduction of such specification can be validated using the CloNe use cases, in particular the epidemic model and the workload generator.

Testbed: We would implement the above mentioned model in the Grid'5000 testbed to generate realistic and flexible workloads and then provide resources dynamically using probabilistic provisioning tools. See `https://www.grid5000.fr/mediawiki/index.php/Grid5000:Home` for a detailed presentation of Grid5000. Some of the initial outcome of this model has been mentioned in Appendix A.3.

5.3.8 Resource Discovery, Embedding and Reconfiguration in CloNe

Bringing together network and cloud computing resources raises several challenges, among which are those related to resource management. It is fundamental to have an integrated view of the existing physical and virtual topologies and characteristics of the resources, as well as the status of all network elements and links. Moreover the provisioning and placement of virtual resources must be done in the best way possible, taking into account the available resources at the time of the request, based on a number of possible criteria from both cloud and network. Further, reconfigurations may be needed either on a periodic basis, (to cater for possible side effects of new virtual resources being instantiated and existing virtual resources being resized or released) or triggered by an unexpected event (e.g. node or link failure).

5.3.8.1 Approach

To address the aforementioned challenges a set of mechanisms able to properly handle resources will be developed. We assume the network operator to be the entity employing those mechanisms, thus the amount of information on the network that the mechanisms can retrieve is quite high.

Resource Discovery (and Monitoring): Today the cloud, i.e. data centres, and the operators network are two distinct domains which CloNe aims at integrating. However there are boundaries that cannot be crossed as these domains will not be willing to share full information about their domain. In our approach we assume to have access to network information such as topology and physical resources as well as the ability to retrieve information on the virtual resources. On the data centers side we do not expect to have such detailed information. We rather expect data centres to provide sets of information (as today e.g. types of VMs it may host) that will allow the algorithms to infer decisions. However, CloNe is not restricted to data centres as it also presupposes the existence of servers scattered along the network. For such servers we expect to be able to retrieve most of its information (e.g. location, physical characteristics, and current load). [35] presents a network discovery algorithm which is being extended in order to also be able to retrieve information about data centers and servers scattered in the network.

Resource Embedding: Virtualised network environments offer a much higher level of dynamicity than traditional networks (e.g. deployment/removal of virtual networks, reallocation of virtual resources, variable load and occupation of resources), which copes well with the cloud computing paradigms. In this sense we are developing an algorithm for mapping CloNe requests based on VNs. However virtualisation is not the only possible network solution as Section 6 highlights, thus we intend to also develop an algorithm in which VPNs are the network service of CloNe. We intend to develop a combined mechanism that will perform balanced decisions taking into account requirements of both network (e.g. latency, bandwidth) and cloud resources (e.g. CPU, storage). Appendix A.4 presents some preliminary work in this area.

Resource Reconfiguration: Due to the dynamics of mobile environments, requirements of users and services, it may be required to reconfigure a request (e.g. user requirements, unexpected situations, load balancing of the cloud or network, business policies). Depending on different situations, actions can be taken at different levels: in the cloud resources, in the network resources, or in both cloud and network resources. Mechanisms will be developed for: extending Cloud resources to other data centres (or scattered servers) or moving cloud resources from one data centre (or scattered servers) to another (using the defined mapping algorithms); creating new network paths and reconfigure existing ones (need for more bandwidth, less latency, failure, load balancing network resources). The algorithms will decide on (1) when to reconfigure and (2) how to reconfigure the network and resources to optimise the previously mentioned parameters.

5.4 Conclusion and Further Steps for Management

The management architecture is aimed at addressing a number of challenges arising from the unification of cloud computing and network virtualisation into cloud networking. The key components of the architecture enable configuration and service requests in terms of higher-level of abstractions, autonomous fault monitoring within individual and multiple virtual layers, and dynamic resource provisioning, migration and optimization for efficient resource usage throughout the cloud. Algorithms running within the architectural framework are focused on being scalable, decentralised and autonomous solutions, allowing for efficient configuration, provisioning and monitoring. Future work will include refinements of the management concepts and the architecture, algorithm development, and proof-of-concepts in terms of simulations as well as prototyping efforts in some of the cloud network management aspects. Algorithm development will to a large part be focused on single-domain problems, whereas further detailing of the architecture for cross-domain purposes will be part of the architecture development.

6 Network View

This section deals with the mapping of the CloNe architecture onto specific networking technologies. In particular, the FNS represents an abstraction of the basic network resource in the CloNe architecture, and we consider how the FNS concept can be instantiated on different types of networks, with focus on existing or near-term protocols and systems. The main features of an FNS that are of particular importance here are efficient provisioning, traffic support, and ease of management. Hence, we consider how different networking technologies could make it possible to quickly set up, modify, and tear down FNSs; how a wide range of different traffic classes could be supported in an efficient way; and how a simplified network view can be provided to the user, abstracting away unnecessary details and providing a sufficiently rich set of operations in order for the user to control the FNS according to his/her needs.

In this section, three main network types are considered: virtual private networks, flow-based networks and fully virtualised networks. The scenario we choose to exemplify the use of CloNe is equivalent to the Data-Centre interconnection use case presented by OConS. However, in this document, we concentrate on the specifics of virtualisation and compare technologies with different levels of maturity to implement the CloNe architecture as a whole, while the OConS approach presented in [36] concentrates on the capabilities the Data Centre interconnection use case requires to be implemented by the underlying communications infrastructure, which in their case happens to be OpenFlow.

6.1 Mapping to Virtual Private Networks

6.1.1 Introduction

Cloud computing offers interesting advantages to enterprises: flexible consumption of resources, minimization of IT investment, reduction of operational costs. However, the uptake of enterprise cloud services and applications will not be possible before obstacles to fulfil enterprise-grade requirements, in terms of reliability, availability, performance and security are overcome. The fulfilment of SLAs is absolutely essential for enterprises to consider migration of computing infrastructure and applications to the cloud. Virtual Private Networks (VPNs) represent the main building block of todays enterprise networks. This is not likely to change very significantly in the foreseeable future. VPNs have gained a strong foothold in service providers networks and a reputation for robustness and reliability amongst enterprises, including small to medium businesses. Whats more, VPNs have been quite successful in alleviating the operational effort of running complex networks by moving this burden to service providers and let enterprises focus on their core business. For these reasons, it is clear that interoperability with VPNs will represent a crucial requirement for any cloud solution to gain widespread acceptance in the enterprise market. The term VPN is usually employed in two different contexts:

- Customer Premise Equipment Based VPNs (CPE-based VPNs): As the name implies, the devices that are involved to set up the VPN are physically located on the customers facilities (or, more precisely, at the tunnel end points), whereas the network plays a mere transport role. From a routing perspective, this tunnel is viewed as a simple point-to-point link. Provisioning and management of the VPN is up to the customer, typically by manual configuration of the tunnels between CPE. In practice, to minimize the customer operational effort, the network service provider may be in charge of provisioning and managing the CPE. The tunnel between CPE is implemented by means of encapsulation techniques such as IPsec.

- Network-based (a.k.a.provider-provisioned) VPNs: in this case, the service provider is in full control of the VPN. All related configuration, operation and control procedures are provided by the service providers network. Customer network is supported by tunnels, which are set up between pairs of edge routers. In most cases, these tunnels are based on MPLS, taking advantage of features such as label stacking to enable tunnel aggregation and scalability. Usually, two basic VPN variants are considered, depending on the protocols on which the VPN is based: L2VPN and L3VPN.

Figure 6.1 highlights the basic approach followed on the two VPN models. CPE-based VPNs are supported by a full-mesh of tunnels between CPE (also called CE), whereas in Network-based VPNs each CPE is connected to a single VPN edge router (PE). In the case of L3 VPNs, each PE hosts as many VRFs (Virtual Routing Function) as VPNs to which it is directly connected. A VRF can be seen as the equivalent of a VPN-specific routing table. Creating a new VPN or modifying an existing VPN usually requires configuring or reconfiguring a specific VRF. In the case of L2 VPNs, PEs support multiple VSI (Virtual Switch Instances), which handle MAC addresses, and essentially mimic the behaviour of a conventional layer 2 switch.

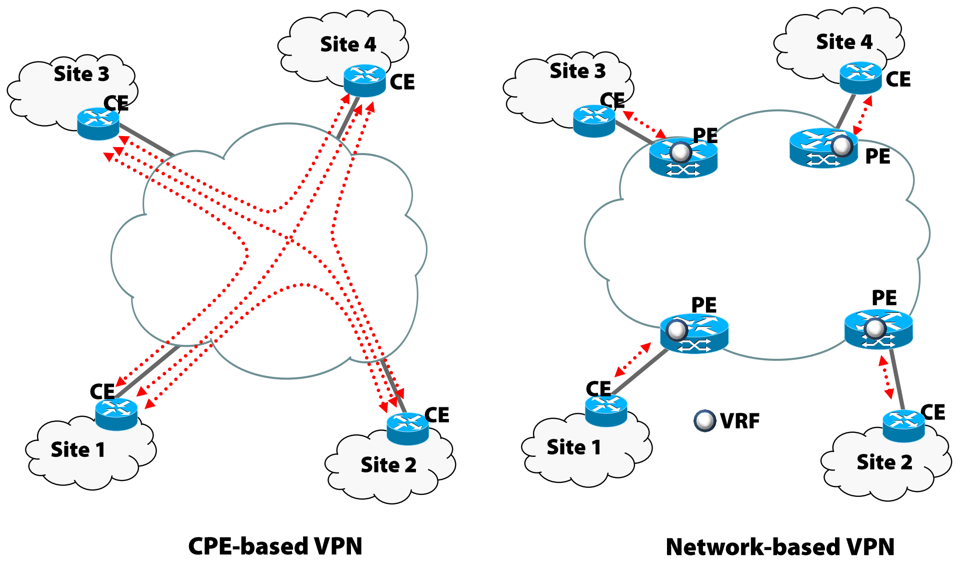

Figure 6.1: Basic VPN types: CPE-based vs. Network-based

6.1.2 VPNs providing access to Clouds

Several scenarios can be defined to access clouds. By default, the network provides a pure connectivity service between end users and cloud resources: this corresponds to what we call transparent access. Since the network service and the cloud service are unaware of each other, no meaningful end-to-end SLA can be put in place; also, any security control, including access control and information confidentiality, has to be enforced by the customer without any intervention of the network in between. In many cases, especially in the consumer sector, this lack of reliability and quality guarantees will probably not represent a major issue for customers. However for enterprises, including small/medium businesses, this has been one of the main reasons for the reluctance to embrace the cloud paradigm. The use of VPNs (both CPE-based VPNs and Network-based VPNs) circumvents many of the problems posed by the transparent access to clouds.

Access through CPE-based VPN In this case, a CPE-based VPN, supported by a specific tunnelling technology (e.g. IPsec, GRE) is used to interconnect the customer premises to the cloud data centre. Similarly to the transparent access case, network and cloud services are separately managed and controlled. However, two important differences are to be noted. Firstly, by using a VPN tunnelling technology supporting encryption (e.g. IPsec), a minimum level of security in terms of confidentiality and access control can be guaranteed. Secondly, because private addressing can now be used in the cloud, those resources can be seen as an extension of the customer private network. This means that migration of computational resources to the cloud becomes seamless. On the flip side, this solution can do very little to guarantee end-to-end performance and reliability, which in practice means that it will not be an option for many enterprises to adopt. On the other hand, a solution based on the separate negotiation of SLAs for cloud and network resources, although possible in theory, would be very difficult to accomplish in practice.

Access through Network-based VPN In this case, the Network-based VPN is able to provide a robust, reliable and secure network path between the customer premises and the cloud resources. It is important to note that a Network-based VPN becomes an integral part of the customer private network, in the sense that the Network-based VPN control plane participates in the routing of the private network. This represents a major advantage because a fully integrated reliable service offer can be put together with Network-based VPNs, in such a way that any reconfiguration of the customer network (including the provisioning of network resources required to accommodate new cloud resources) is appropriately accommodated in the VPN offered by the service provider. Examples will be provided in the following section.

6.1.3 Challenges to use VPNs in a Cloud environment

A number of issues have to be sorted out before the access of enterprises to cloud services can be supported by VPNs. On the one hand, current commercial cloud platforms are ill prepared to handle the requirements of enterprise customers [37]. Conversely, the dominant VPN models were not conceived to deal with characteristics of the clouds. By definition, cloud resources may be instantiated or reconfigured by the customer on a self-provisioning basis. This contrasts with the traditional VPN provisioning approach, which typically follows a rigid workflow, based on human intervention by the service provider. Elasticity of resources, on-demand reconfiguration, resource mobility have never been requirements in VPN environments, as VPNs are expected to be a relatively stable service offering, with relatively infrequent configuration changes (e.g. adding new VPN sites, changing bandwidth capacity).

In this section two cases of dynamic cloud service instantiation are provided for illustrative purposes. The first example, depicted on Figure 6.2, corresponds to the instantiation of new cloud resources, (for example, as a result of a sudden increase of computing capacity needs that cannot be fulfilled in the local data centre), usually known as cloud bursting. Assuming that no resources had been previously provisioned to that customer by the cloud provider, the first step would be the establishment of a subnet on which the new resources would be accommodated. This would imply the creation of a new VRF in the corresponding edge router. Subsequently, the newly created subnet would be announced to the remaining edge routers supporting the same VPN, which in turn would forward this information to the local area networks in the enterprise sites.

The traffic admitted from the cloud provider into the network is controlled at the corresponding

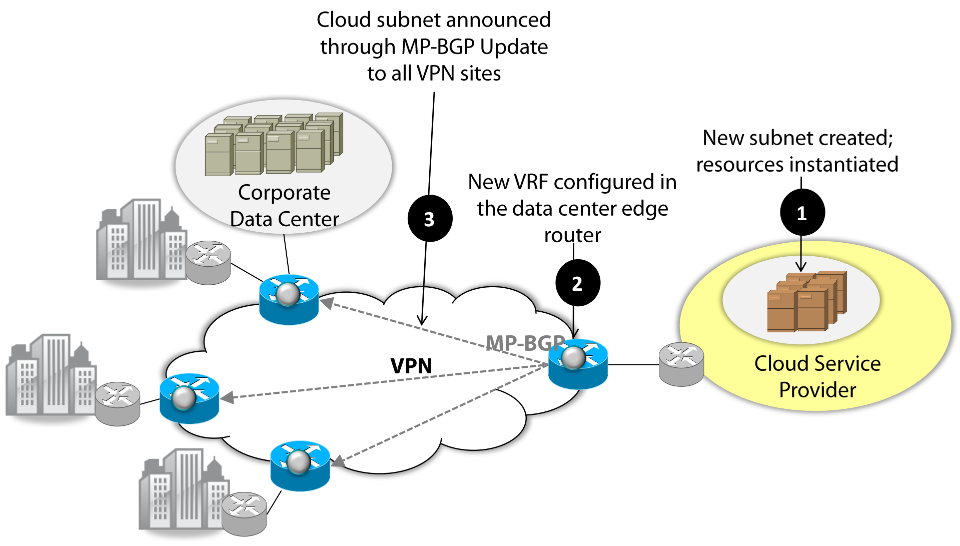
Cloud subnet announced through MP-BGP Update to all VPN sites
New subnet created; resources instantiated
New VRF configured in the data center edge router
Corporate Data Center
3
2
1
MP-BGP
VPN
Cloud Service Provider

Figure 6.2: Cloudbursting scenario

ingress point (i.e. edge router). Thus, for any increase or decrease of this capacity, a corresponding reconfiguration of the VPN edge router might be necessary. The second example corresponds to the case where computational resources (including the corresponding subnet) are migrated from the enterprise data centre to the infrastructure managed by the cloud provider. Again, a new subnet would be provisioned (assuming that it had not been provisioned before), which would require the setup of a new VRF in the respective edge router. The migration of the corresponding subnet from the enterprise data centre to the cloud data centre would be announced to the remaining edge routers, similarly to the first example (except that the removal of the subnet from the enterprise data centre would also be announced). In the case of L2 VPNs, mobility of resources have somewhat different implications because, contrary to L3 addresses, L2 addresses are typically not bound to physical location.

6.1.4 A new VPN abstraction

Contrary to other forms of network virtualisation, VPNs do not enable a clear separation between virtual resources and infrastructure. This means that the VPN resources cannot be easily controlled and administered by any entity other than the VPN service provider itself. This also means that from the customer perspective, a VPN is essentially a black box, of which only the access points are visible to the customer. Any VPN-related protocol that runs inside the service provider administrative domain is not visible to the customer.

The sections above have shown that cloud services, particularly the requirements posed by self-provisioning and resource elasticity, bring new challenges to traditional VPN provisioning and management mechanisms. Thus, a new VPN abstraction, which can provide the customer with limited provisioning capabilities, is needed.

To some extent, the abstraction provided to the customer by a VPN is similar to a router, for

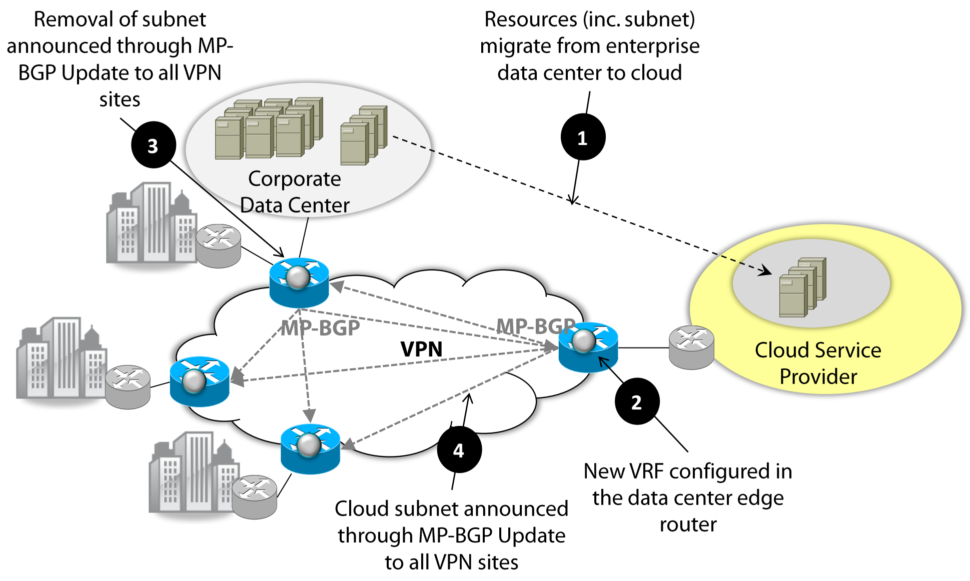
Removal of subnet announced through MP-BGP Update to all VPN sites
Resources (inc. subnet) migrate from enterprise data center to cloud
3
1
Corporate Data Center
MP-BGP
MP-BGP
VPN
Cloud Service Provider
2
4
New VRF configured in the data center edge router
Cloud subnet announced through MP-BGP Update to all VPN sites

Figure 6.3: Resource migration scenario

L3 VPNs, or a switch, for L2 VPNs. Each customer network interface is attached to a specific VPN access point and the VPN is viewed a single network piece of equipment (either a router or a switch) distributed across a wide area network. However, in the traditional VPN model, the customer has no possibility to directly interact with the network. Any kind of reconfiguration of the VPN service has to be executed through the service provider.

The main challenge is the redefinition of the traditionally rigid boundaries between customer and service provider, in such a way that a limited set of network functions can be controlled by the customer, either directly, or as a result of a reconfiguration of computing resources.

The single-router abstraction, initially advocated in [18], sounds like a promising solution to overcome these problems. Basically, a single-router abstraction would provide the means to define a limited set of functions to be controlled by the customer (provisioning and reconfiguration of cloud resources, definition of QoS and security policies, etc), whereas the VPN internals (e.g. routing protocols) would still remain hidden from the customer view, in order to keep integrity of the network resources and minimize security issues. In order to accommodate L2 VPN services, in addition to L3 services, we propose to generalize the single-router abstraction to single-node abstraction. It is clear that the practical materialization of this concept has a few limitations that should be taken into account. One important difference of the VPN router/switch is that in the access bandwidth may vary in small steps due to the cost of connectivity, whereas for physical on-site routers/switches, symmetrical 1G on the router/switch is the norm, with possible alternatives of 10/100M or 10G.

6.2 Mapping to Flow-Based Networks

This section explores how Flash Network Slice are instantiated, created and mapped to networks relying on flow-based networking, with special focus on using OpenFlow technologies to dynamically deploy and adapt FNSs according to network conditions and user requirements.

Figure 6.4 depicts an example with two FNSs deployed over cloud platforms and an OpenFlow substrate. Each slice is composed of virtual nodes, provided by cloud providers, and virtual paths offered by network providers and supported by OpenFlow switches.

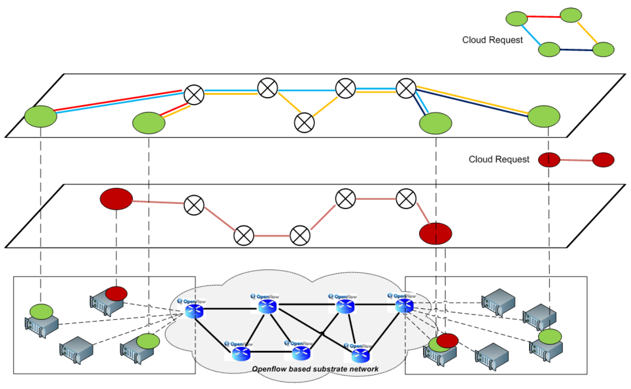

Figure 6.4: FNS mapping to OpenFlow

The FNS request is composed of a set of virtual nodes (i.e. cloud resources) interconnected via a set of virtual links across a shared substrate network. The FNS resources are jointly allocated by the cloud providers and network providers. The FNS request is split into smaller graphs according to specified optimisation objectives and constraints. Each subgraph is mapped to the appropriate cloud platforms. In Figure 6.4 this corresponds to the two data centres but in a more general case the mapping can involve also the provider network resources and services. To establish the inter-subgraph links between the involved clouds across the substrate network, OpenFlow technology is used to map the FNS paths to the shared substrate network.

We consider two main approaches for mapping FNSs onto OpenFlow networks, which are further outlined in Section 6.2.2 and 6.2.3 below:

1. Through intermediate virtualisation controllers, such as FlowVisor.

2. Through dedicated controllers specifically designed for control and management of FNSs.

6.2.1 OpenFlow Background

OpenFlow is an open standard that enables running experimental or new protocols in existing production networks. It relies on a control and forwarding plane separation paradigm to enable flow-level control and management of switches. The data path of an OpenFlow switch is composed of a flow table with actions associated with each flow table entry. Users control flows by installing appropriate actions in the switches. OpenFlow also provides an open protocol to allow an external *controller* to communicate with the OpenFlow switches and control their flow tables. There are several initiatives for developing OpenFlow controller software, including NOX[1], Maestro[2], Beacon[3], Trema[4], and ONIX[5].

OpenFlow has several characteristics that makes it an interesting candidate for CloNe:

[1]`http://noxrepo.org`
[2]`http://code.google.com/p/maestro-platform/`
[3]`http://www.openflowhub.org`
[4]`https://github.com/trema`
[5]`https://www.usenix.org/events/osdi10/tech/full_papers/Koponen.pdf`

- The simplified physical switching infrastructure, which facilitates hardware virtualisation.
- Potential for online modification, migration and backup of network controllers.
- The integration with existing technologies and the development of standalone technologies.
- More dynamic and flexible approach to a network resource than traditional technologies.

There are two main options for routing flows within an OpenFlow network:

1. Native OpenFlow forwarding: flow table entries are installed for the specific header fields representing the traffic within the FNS. In this case the selected fields for the matching must be unique in the entire network, which means that the controller needs to perform conflict resolution to ensure FNS isolation.
2. Tunnelling: Tunnels are between the different ingress and egress points with one of the tunnel technologies supported by OpenFlow, and the packets are encapsulated and decapsulated accordingly. This will eliminate the need for conflict management, and will also reduce the size of the flow tables in the switches. As of OpenFlow v1.1, there is support for two protocols that provide this functionality: MPLS and QinQ VLAN.

Another consideration is how the paths are initialized and deleted. Currently, there are two ways to instantiate paths relying on the OpenFlow rules:

- Permanent path deployment: OpenFlow rules are injected in the flow tables as permanent entries. This approach potentially suffers from scalability limitations, since entries in the flow table will never be removed and this caps the number of deployed flows to the maximum possible number of entries.
- Online path deployment: OpenFlow rules are injected in the flow tables on demand, as temporary entries. When a switch cannot find a flow table entry that matches an incoming packet, the switch forwards the packet to the controller to dynamically compute the path ”online” and determine the appropriate policy to be applied and injected. These operations increase the delay for flow setup, but could improve scalability when combined with a suitable replacement policy for flow table entries.

6.2.2 Intermediate Virtualisation Controller

A possible approach for FNS creation is the use of an intermediate virtualisation controller, such as FlowVisor, which acts as a transparent proxy server between the switches and the corresponding controllers. FlowVisor slices an OpenFlow switch into several virtual switches, thereby making it possible to control virtual OpenFlow switches from regular controllers, such as NOX. Figure 6.5 illustrates how we could use FlowVisor to create slices in the single router/switch abstraction using Openflow Enabled Switches.

Let us consider three network slices, orange, green and blue; the FlowVisor node works as a proxy between the OpenFlow enabled switches and the slice controllers (e.g. NOX controller) and is responsible for slice creation and rule management. For each slice, there is a controller responsible for control of the flows within that slice. Suppose that a host in the blue slice starts a new flow and sends a packet through switch sw2; this should trigger a request to the appropriate controller what to do with the packet (unless rules are permanently deployed for the flow by the controller), so sw2 will connect to the FlowVisor node which in turn will direct it to controller 1, in the same way controller1 will send a response to sw2. Hence, the FlowVisor node maintains the rules that

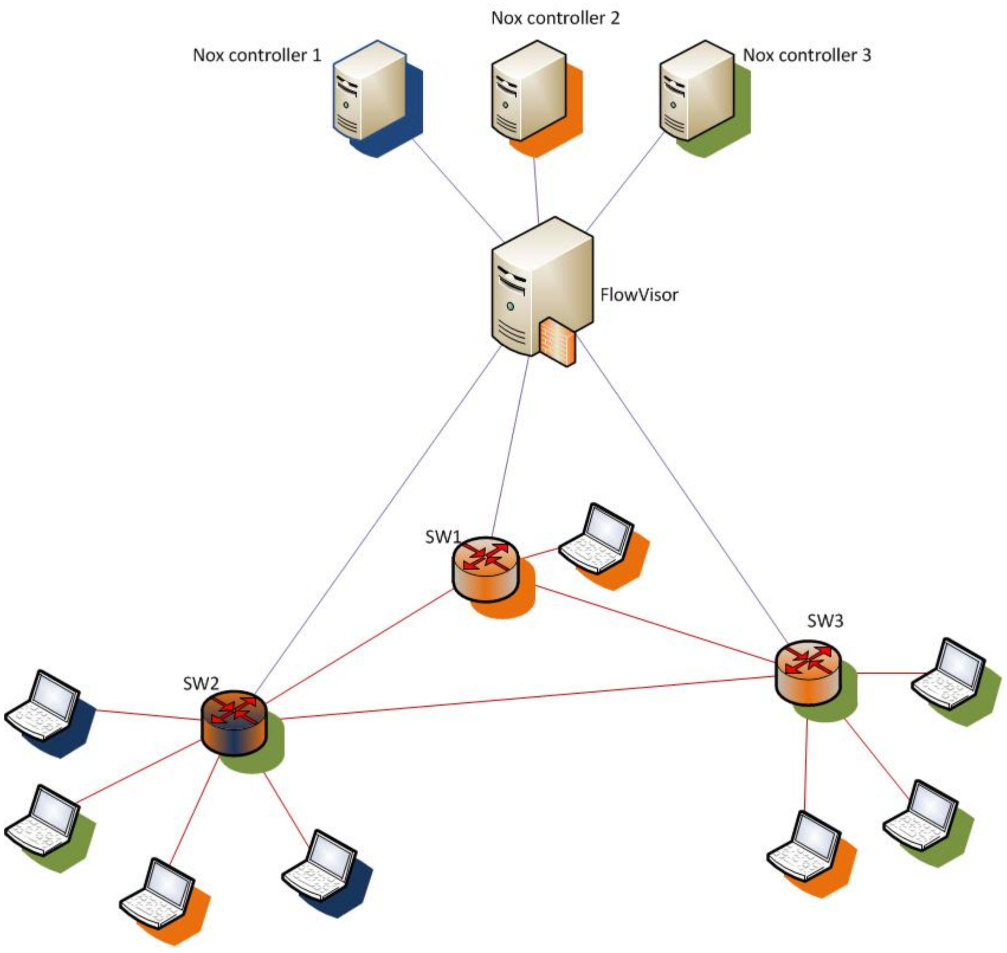

Figure 6.5: Slicing using FlowVisor

define the network slices, manages the connection between slices and controllers, and guarantees isolation between slices.

This model allows different slices to use different network models and routing paradigms: single switch, single router, etc. On-demand creation and modification of FNS can be achieved by changing the FlowVisor configuration. For example, from the infrastructure provider point of view, it would be achived by only changing the slice's corresponding port in the FlowVisor, this can convert it to a totally different slice in terms of functionality and topology. Convergence is guaranteed by the lifetime of the rules installed in the OpenFlow switches. Changing the slice topology would be more challenging because the infrastructure provider may have a pool of controllers ready to connect to a specific slice with minimum reconfiguration, but will not have prebuilt slices for every possible requested FNS, so an efficient mapping between FNS and the FlowVisor slices is needed.

All the controllers and a FlowVisor could be integrated into one server; this server will be responsible of slice creation and flow management. The use of a centralized FlowVisor may have many advantages but it also constitutes a single point of failure. This could be addressed through redundancy techniques, such as a fully distributed FlowVisor design, or by holding the existing flows in the switches while the backup server is up and fully synchronized.

6.2.3 Dedicated Controllers

A second approach is the use of a dedicated controller for FNSs (Figure 6.6), that is, a controller implementing the FNS abstraction, and that supports dynamic creation, modification, and termination of Flash Network Slices.

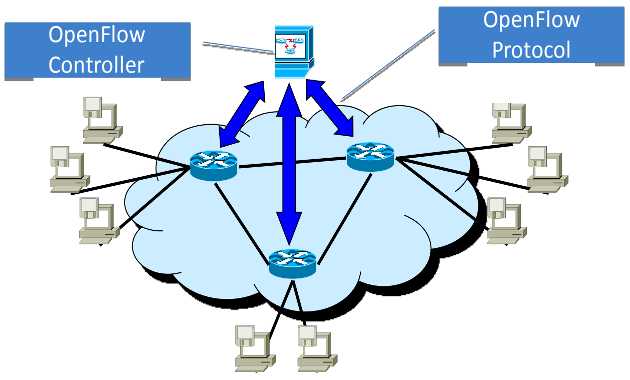

Figure 6.6: OpenFlow view

6.2.3.1 FNS Controller as a NOX Module

The NOX controller provides a programmatic interface to support advanced control and management functionality based on events, a name space, and a network view. It is implemented in C++ and Python, and has a modular structure that allows users to add their own network control software.

The NOX's network view maintains the topology of the entire substrate network. To ensure slice isolation to restrict the view and the scope of each slice, CloNe is:

- developing a new NOX module called "FNS control module" responsible for establishing and instantiating FNS paths;
- creating a flash network view (FNS view) derived from the global NOX's network view to maintain network topology and slice information.

Once the FNS nodes are mapped to the Cloud platforms (for Figure 6.4, the two data centers), the Cloud providers send the node mapping information to the NOX controller. The FNS control module in the NOX controller extracts the node identifiers (e.g. name or address) and determines the set of OpenFlow switches directly connected to the virtual nodes. Relying on the network view, the FNS control module computes the shortest paths between the virtual nodes while satisfying the user specified link requirements. Next, the FNS control module determines the appropriate OpenFlow policies and rules required to set up the computed paths.

To reduce the delay required to set up the FNS, the paths are precomputed based on the network view and stored in a database located in the NOX controller called "FNS view" database. This database maintains information about the slice network topology and constraints. This solution would be an extension of PCE [38] based inter-domain solutions. Likewise, inter-domain communications between controllers would be a similar approach to the implementation of pce in a (G)MPLS network [39].

6.3 Mapping to Virtual Networks

In this section, we briefly describe how to materialise flash network slices using the network virtualisation technology, as well as the deployment and management challenges using this technology.

6.3.1 Background

The merits of network virtualisation have been mainly promoted by research projects in the last few years, particularly as a key enabler of new Internet approaches that could overcome the current limitations caused by Internet ossification.

In general, network virtualisation involves the partitioning of a physical network infrastructure into a virtualised substrate (see Figure 6.7). The virtualised substrate consists of virtualised resources–virtual links and virtual nodes—which are combined into virtual networks. The process of mapping (or embedding) a virtual network is based on user requirements, which can be expressed in terms of capacity of virtual links and virtual nodes. A virtual network is created from the substrate in such a way that the requirements are met.

Network Virtualisation enables all network resources, i.e., nodes and links, to be fully virtualised. Through virtualisation, operators could easily deploy different network architectures and protocols, using a single infrastructure. Just like data centre virtualisation, network virtualisation can potentially enable flexibility and dynamism by decoupling networks from infrastructure. Making the network infrastructure capable of matching the dynamism of the cloud would be an obvious advantage for service providers, in order to build seamless end-to-end elastic and agile cloud services. The concept of network virtualisation is not new: as described before, network-based VPNs are essentially separate networks sharing a common infrastructure and can also be seen as a materialization of this idea. However, VPNs cannot be decoupled from the underlying infrastructure and should be seen more as a service, rather than a real network. Abundant examples of network virtualisation are available already today in network operators infrastructure, either in the form of link virtualisation or, to a lesser degree, in the form of network node virtualisation (examples of high-end commercial equipment by major vendors supporting router virtualisation can be found).

An important advantage of network virtualisation compared to traditional VPN approaches is the fact that, because virtual networks are fully separated from the infrastructure, resources can be moved and reconfigured in a much more flexible way. This means that the issues described above concerning migration of resources in VPN scenarios should be handled in a much more straightforward way with fully virtualised networks.

Although very attractive from several standpoints, network virtualisation contains several challenges [40]. Virtualizing the network resources is one of them; ideally the user should have a large degree of freedom when it comes to choosing routing and forwarding paradigms, and this is currently not supported on existing commercial platforms. Research on efficient software virtual routers that can compete with the current hardware routers is under way and promising results have been attained [41, 42]. Other fundamental issues are related to management, monitoring and embedding of virtual networks in the physical infrastructure. While some of these problems have been widely addressed in literature, the requirements and constraints posed by commercial operational environments have not been fully evaluated. So far, network virtualisation and real time on-demand network provisioning have been successfully demonstrated in small-scale research testbeds, but it is clear that there is still a way to go before the technology can be considered mature enough for large-scale deployment.

6.3.2 Implementation

Node virtualisation and link virtualisation are the two basic building blocks of network virtualisation. Node virtualisation has been available in commercial hardware products, in the sense that vendors enable their routers with the possibility to be partitioned into multiple virtual (or logical) routers that can be configured and managed independently. A virtual router/switch should appear from all points of view (e.g. configuration, management, monitoring, troubleshooting) as a dedicated physical counterpart. Equipment vendors have long supported limited forms of virtualisation to support L3/L2 VPNs, e.g. Virtual Routing and Forwarding (VRF), Virtual Switching Instance (VSI), which can be seen as predecessors of full-blown network virtualisation. Currently, several network vendors offer equipment models that support full virtualisation. The combination of virtualisation with routing software (e.g. Linux-based XORP, Quagga routing suites), provides a straightforward solution to set up virtual networks using commodity hardware. This has usually been the approach taken to build network virtualisation testbeds for research purposes.

The other basic component of network virtualisation is link virtualisation, which typically can be achieved through a wide range of alternatives, either based on layer 1 multiplexing, such as WDM, TDM, CDMA, etc, or through data link layer virtualisation using Ethernet VLAN, ATM, Frame Relay or MPLS.

In order to deploy and manage FNS using network virtualisation technologies, updated knowledge of the utilization of network resources is required. This can be achieved by using discovery algorithms that periodically or on a trigger basis inform the administrative domain about the state of the physical and virtual resources, as described in [35].

Secondly, efficient embedding of the FNS on the physical network is needed, which can be achieved by using heuristic greedy algorithms, as described in [43, 44, 45, 46, 47, 48] or through distributed embedding algorithms, as proposed in [49].

Finally, each administrative domain needs to perform resource provisioning on the concerned elements. In the Architecture and Design for the Future Internet (4WARD) project [2], scalable and dynamic provisioning, as well as management of virtual networks have been addressed. The signaling mechanisms and control interfaces defined in the 4WARD project for virtual networks can be applied as well in FNSs. Figure 6.7 represents the 4WARD layered vision of network virtualisation. The top layer represents the physical infrastructure. By virtualising the physical resources and partitioning them into slices, the virtualised substrate can be built in the middle layer. The bottom layer corresponds to the virtual networks which materialize each FNS. Full virtualisation of the network resources, as depicted in Figure 6.7, enables two major features that can be explored in a cloud networking scenario - the first is the integrated management and control of computational and network resources, which now become a single set of virtual resources; the second is the possibility to clearly decouple flash network slices from the infrastructure, thus enabling the implementation of protocols at the FNS level that can be tailored according to the specific characteristics of the cloud networking environment.

6.4 Network View Summary

Table 6.1 tries to summarize the three different network technologies according to their characteristics. It does not intend to compare them, since the level of maturity and the scope of each of these network technologies is quite different. However, it can help deciding a migration path as proposed by CloNe, starting with existing virtual private networks and moving towards forward looking full network virtualisation solutions like 4WARD.

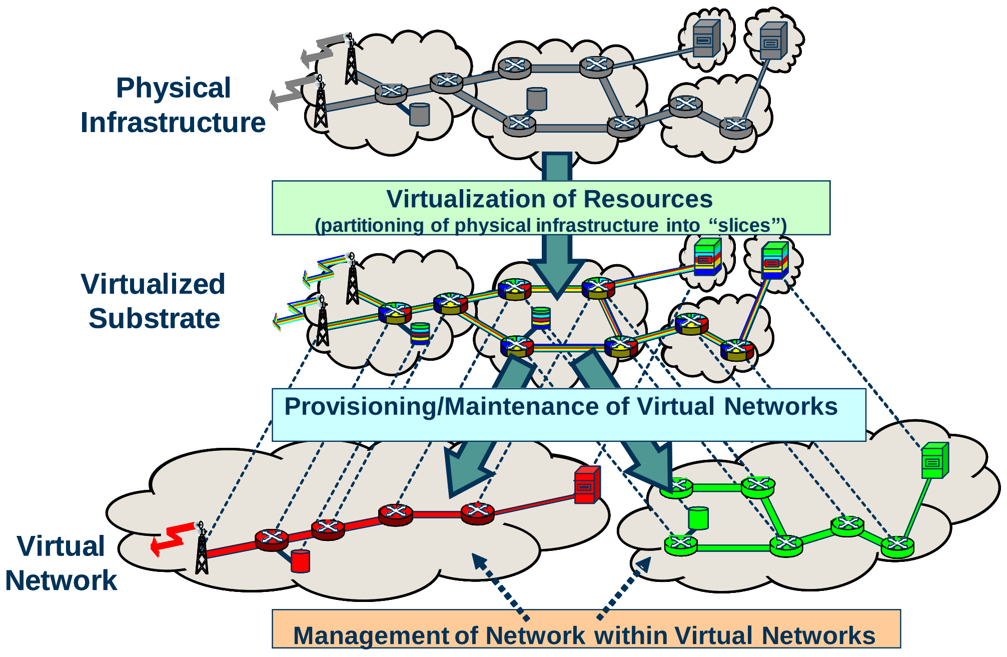
Physical Infrastructure
Virtualization of Resources
(partitioning of physical infrastructure into "slices")
Virtualized Substrate
Provisioning/Maintenance of Virtual Networks
Virtual Network
Management of Network within Virtual Networks

Figure 6.7: Network Virtualisation technology overview within 4WARD [2]

Table 6.1: Comparison of different network technologies.

Characteristics	Virtual Private Networks	OpenFlow	Virtual Networks
Technology Time	+10 years (mature) Deployed	4 years (new) Experimental Deployment and test phase.	3 years (new) Research phase Prototyping
Availability	Up and Running Widely deployed in network providers. Commercial solutions available, e.g., Cisco, Juniper.	First steps at universities and vendors.	Future internet research Projects, e.g., 4WARD[2].
Scalability	High Some issues on the BGP tables for large scale operators[50].	Medium Centralized approach[51], i.e., controller.	Low Limited number of virtual nodes per physical node.
Flexibility	Low	Medium Separation of the control plane from the data plane. Different flow tables per controller.	High Dynamic provisioning of virtual resources. Migration of nodes and links in real-time.
Programmability	None Black boxes	Medium Only on the flows	High Nodes and links can be modified according to the needs.
Provisioning Time	Ranging from minutes to hours With automated provisioning	Ranging from miliseconds to seconds Time to setup the flows	Ranging from seconds to minutes Time to provisioning the virtual nodes
Protocol Stack	MAC and IP layer, i.e., L2/L3	Cross Layer Technology Rules can be applied from MAC layer to application	Layer independent technology Virtualisation techniques can be applied in all layers. From the physical layer, i.e. optical wavelength, to the application layer.
Standards	RFC 2341 - IP Based VPNs RFC 2547 - BGP/MPLS VPNs Other RFCs on the security side [52]	OpenFlow *de facto* standard [53] Open Networking Foundation [54]	Initial standardization steps at: VNRG IRTF [55] FGFN ITU-T [56]

7 Security Architecture

Cloud networking introduces new security challenges affecting availability, integrity, confidentiality, authenticity, and privacy. In a first step we show the security goals for CloNe, relevant attackers, and based on this identify relevant security challenges for CloNe (see Section 7.1). To address these challenges we define a security architecture in Section 7.2, introduce security parameters (Section 7.3), show the roles and responsibilities in the architecture (Section 7.4), and show the relation to the technical architecture (Section 7.5). In Section 7.6 we show an example for a security goal translation. Further steps for the security work for CloNe is given in Section 7.8.

7.1 Security Analysis and Requirements

Before defining the security architecture we have to define the security requirements which are based on a security analysis. The security analysis in this section consists of a definition of CloNe relevant security goals and relevant attackers in CloNe. Basis of this analysis it the preliminary architecture and use cases of CloNe. As a result of this security analysis we show security challenge that are introduced by CloNe. How to address these security challenges is shown in the rest of this chapter.

7.1.1 Security Goals

Availability Availability means, that a subject is not able to impact the ability of a system to deliver services to its users in an unauthorised way. Availability is generally a quantitative metric against measurable parameters (e.g. number of users serviced in parallel, network bandwidth, response time). It has strong ties to (perceived) quality of service.

In the case of cloud networking this means that no user without administrative privileges on the cloud networking infrastructure is able to impact the service of the other users.

Integrity Integrity means, that a subject is not able to alter secured data without authorisation; one instantiation of this is that all modifications can be detected after the fact, for example using electronic signature schemes.

For cloud networking integrity of data that is stored on the cloud networking infrastructure is important. Also 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 send to, from, or inside the cloud networking infrastructure.

Confidentiality Confidentiality means, that no one is able to access data without authorisation. This typically requires that users possess the right credentials, such as encryption keys. This requires that the appropriate tools for managing these credentials (distributing, verifying and revoking) are included in the management infrastructure.

Similar to integrity this means confidentiality of both, the data stored inside the cloud networking infrastructure, as well as the data that is contained in the communication.

Authenticity Authenticity means, that one can proof that someone or something is the one/thing that it claims to be.

Authenticity is needed for the cloud networking infrastructure so that a user can verify that he is communicating with the correct infrastructure. Also users can authenticate their identity in order to access the infrastructure.

Non-Repudiation Non-Repudiation means, that a subject who performed an action is not able to disclaim it afterwards.

For cloud networking this is strongly related to traceability, i.e., to verify where your virtual infrastructure is located and if it is conform to the agreed policies. Non-Repudiation is also important for accounting.

Privacy Privacy means, that a subject is able to decide which personal information it wants to reveal. Anonymity, i.e., to hide the subject's identity in a set of other identities (anonymity set), and pseudonymity, i.e., the use of pseudonyms instead of real names, are ways to enforce privacy.

Privacy is always a trade-off between information that is necessarily needed to provide a service and the user, who wants to provide as little as possible personal information. In cloud networking first the needed information has to be identified and additionally it has to be ensured the legal directives of the country where the physical infrastructure is located are followed as the virtual infrastructure can pass legal borders.

7.1.2 Attacker Model

The following gives an overview on roles and capabilities that an attacker might have. This attacker model will be used when designing the security architecture for CloNe networking.

The attacker model bases on an external attacker that tries to access resources on the cloud infrastructure. To do this he can eavesdrop incoming and outgoing communication of the cloud networking infrastructure and try to get access to the infrastructure itself, e.g., by using vulnerabilities of the system. For some scenarios, an internal attacker might also be of interest, e.g., an employee of the cloud networking provider that accesses customers' data. A similar attack might be a supplier that introduces trapdoors in hard- or software in order to access data that is processed on the infrastructure. Additionally, the attacker might be a legitimate user of the cloud networking infrastructure and uses this access, e.g., to attack other users' data.

External and internal attackers are also often used for analysing cloud computing. In the cloud networking case legal aspects and legal intercepts have to be covered additionally. Due to the fact that virtual components can move to arbitrary physical cloud networking infrastructures they might pass legal borders. Besides the fact that legal intercepts are not classical attacks they might violate security goals of the cloud networking customers. Therefore the physical location (legal space) has to be considered when distributing virtual components.

7.1.3 Resulting Security Challenges

Information security The main challenge from information security side is that the customer processes or stores data in the CloNe infrastructure. Data that is stored in the CloNe infrastructure can be encrypted to ensure its confidentiality and integrity. In the case of processing data there is no mechanism there that guarantees confidentiality and integrity. In this case the customer has to trust the operator. In case of encrypted communication keys have to be exchanged between the communicating parties. As the infrastructure might change dynamically also a dynamic key distribution infrastructure is required.

With security policies a customer can define how its data should be handled. On the other hand an operator defines security policies on which basis security functionality is integrated into his CloNe infrastructure. The challenge is where to place policy decision and policy enforcement point into the architecture so that the policies of the customer are followed. A special challenge is how to detect and react on changes in CloNe operators security policies.

Virtualisation management The management of CloNe includes the access to physical infrastructure. How to implement this access and delegate the access if virtual infrastructure is moved in CloNe is a challenge.

Isolation In CloNe the physical infrastructure for communication, storage, and processing is shared by different customers. Besides the separation of communication and the separation done by a Hypervisor the CloNe management has to take care of following SLAs of all customers.

Misuse protection In the same way as it happened already for pure cloud computing the CloNe infrastructure can be misused, e.g., for spamming. Mechanisms for detecting such misuse must be taken in place.

DoS protection The CloNe infrastructure must be resistant against Denial of Service (DoS) attacks. Because of the dynamic, distributed, scalable, and flexible nature of CloNe these attacks might have a higher impact compared to the same attacks on cloud computing. Mechanisms for preventing and detecting DoS attacks should be introduced.

7.2 Security Methodology

Cloud network, whether operated by a single or multiple operators, requires quantifiable security levels. This would ensure that all the actors (application providers, enterprises, business partners, end-users, data centre operators, cloud site operators, and network operators) obtain a better view of the current security grade of the network and are aided in managing their individual application scenarios, business models, usage, storage, and migration policies besides recognizing plausible operational and management risks.

The overall security requirements can be described using a set of parametrizable security goals / objectives. Each security goal must be characterized with respect to the underlying resource(s), and is therefore used to constrain the security parameter(s) of resource(s). It is imperative to start off with parametrizable goals, which can be finally mapped to configurations of underlying resources. For our purpose, we propose to use the terms goals and objectives as synonyms.

From the user side: For each goal, we would have a set of security parameters, for example geographic location, integrated security through Software Development Life Cycle (SDLC) etc. These security parameters may then be mapped on to resource constraints, which describe the resources being constrained to specific values (or ranges), in order to satisfy the respective parameters.

From the operator side: For each security parameter (which has defined the resource constraints), the operator must select its security mechanism(s), which would be used to implement the desired resource characteristics specified by the user. These mechanisms comprise of security services which help fulfil the requirements of the individual parameters, i.e., the resource constraints.

Once the parameters are constrained to obtain configuration of resources, it would help execute the security parameters, which would help fulfil the respective goals. These steps are performed before virtual resources are moved to an operator side, i.e., before initially placing the virtual resources in CloNe and when moving the virtual resources inside CloNe from one side to another. This process is initiated from the DCP and handed over to the control plane of a single domain.

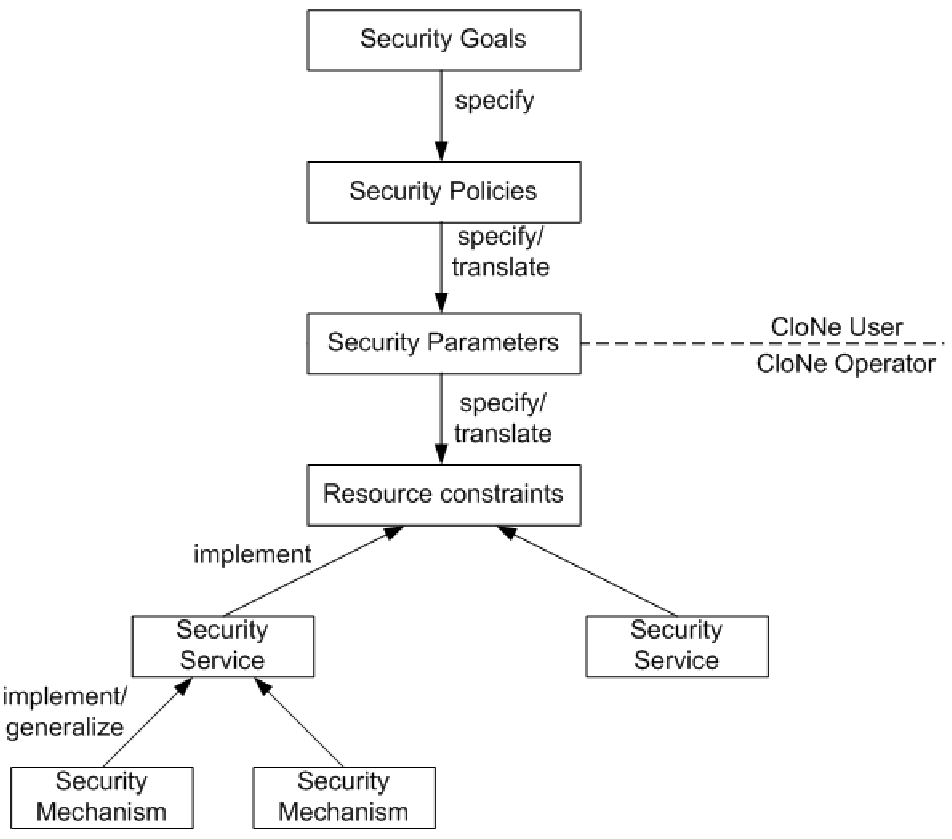

Figure 7.1: Security mechanisms

The user's access to all resources needs to be defined by access control policies (see Section 7.7). Favourably, mechanisms for anonymised access to resources is supported by the access control.

The following Figure 7.1 elucidates the hierarchical structure between security goals, security parameters and the resource constraints from both the CloNe user and operators side.

The security framework follows a modular approach to reuse the goals and their translations to the underlying resources. The modules are realized as sub-goals, or security parameters which can be translated into a group of resource constraints. For example, two separate goals, i.e., "Ensure resources are Payment Card Industry Data Security Standard (PCI DSS) compliant" or "Ensure that there is a minimum security level for the involved metrics" might be entirely different, but share a large number of common parameters (sub-goals). Instead of translating the entire goal from scratch, the security goal translation function will have the ability to reuse the modules that have already been translated. Therefore, it would promote scalability and reduce overall complexity. The following subsection describes how the security goals are obtained, both from the user and generated on-the-fly to ease the overall goal translation process.

7.2.1 Obtaining Security Goals

Security goals must primarily be obtained from the customer through an SLA at the DCP. The underlying resource set is dynamic throughout the lifecycle of a requested service, and hence extra care should be taken to keep the goals consistent, irrespective of the implementation of the requested service and utilization of the underlying resources. Security is usually a non-negotiable characteristic and the goals must not be reformulated unless explicitly specified by the tenant or the administrator of the individual domains.

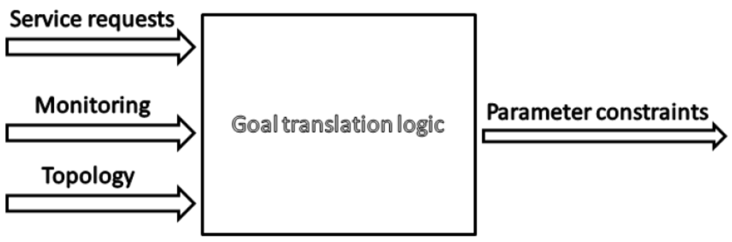

Figure 7.2: Security goal translation

However, it might be possible that the existing goals specified at a specific architecture layer, might not map directly to the security parameters/sub-goals to be implemented at lower layer(s). Therefore, it is imperative that the lower layers must have additional mechanisms to specify security goals, specific for the resources at that certain layer. For example, if a security goal specified at the distributed control plane doesn't map directly to the security parameters/sub-goal to be implemented at the cross-domain infrastructure layer, a mechanism should be realized to specify additional security goals specific for the cross-domain layer to implement the desired parameters/sub-goals.

Finally, monitoring procedures have to be put in place to obtain a feedback about the status of the resources. These measurements, carried out at the VM or physical layers shall assist the security goal translation function to release constraints to the resource management function. The resource management function will then enforce the constraints on the resources. The feedback information might also be useful for the user, who might want to reformulate the goals after receiving undesired performance results.

7.2.2 Security Goal Translation

The security goal translation process includes receiving a set of inputs which need to be provided to the overall goal translation function. The overall goal translation function translates the goal(s) in order to generate security parameter(s). These security parameters are used to specify the security characteristics (parameter constraints) of the underlying resources.

The inputs required by the security goal translation function for generating the overall parameter constraints include, but are not limited to:

1. Service requests: The user shall specify the desired behaviour and properties of the requested service through a well-defined user interface. Ideally the constraints described in the service request must map fairly directly to the underlying resource set. Moreover, the tenant shall have the option to modify their requests within the life-time of the agreement of the service request.

2. Monitoring: The measurements carried out at different architecture layers provide a clear picture of the overall functioning of the infrastructure components, and are essential to gauge the status of the resources.

3. Topology: The goals can't be translated to security services (and further to security parameter constraints) unless the function is aware of the underlying network topology.

The goal translation function as shown in Figure 7.2 should be generic enough to be implemented at any distinct layer, as goals may be specified at any layer, and not only at the DCP. Therefore, its design should be implementable at both the single, as well as the cross-domain infrastructure layers.

Moreover, as the security goal translation function requires input from different management functions, it should have an overall generic interface and should use a widely accepted communication mechanism. This would ensure communication between different parameters lying in the same, or different administrative domains.

Finally, it would be safe to state that the security goal translation function is in actuality a subset of the overall goal translation function, which is described in Section 5.2.1. It would be based on the same structure as the overall goal translation function, and would be focusing on generating the security specific configurations of underlying resources for facilitating secure resource management.

7.2.3 Auditing Mechanism

It is equally important to invoke auditing procedures that will check whether the parameter constraints are being fulfilled or not. The auditing mechanism shall work in a modular fashion, with the respective modules being invoked at repeated intervals. Each module shall audit, assert and assure specific sections of the infrastructure and will have its respective invoking frequency.

An auditing mechanism shall be chosen on the ease of integration into the existing security framework and its ability to be platform and programming language agnostic.

7.3 Security Parameters

This section provides initial listing of the high-level security parameters that may be constrained, in order to attain a quantifiable security level for a cloud network. The parameters are included with only a high-level description and will be modified/explored as further progress and discussions are carried out. The exact set of resources that will be characterized by each parameter is still open and shall depend upon the overall architecture.

Ideally, the security parameters must be self-contained and work independently from each other, unless they delegate responsibilities between themselves, or exchange information. This step is imperative as it will guarantee that the overall security framework behaves in a modular fashion and the non-functioning of one component, will not guarantee the failure of the surrounding components.

The modules of the framework must be easy to integrate with each other, to allow complete flexibility and adaptability. These modules might then be easily modified or integrated as per the SLA or requirements set in by the administrator. Some of the security parameters include geographic location, security compliance framework, security levels of individual metrics, integrated security through the SDLC and resource sharing policy of the cloud.

7.4 Roles and Responsibilities

There are three major, concrete roles discussed in Section 2 that have been identified within the cloud networking architecture scenario. These are:

1. Administrator: It has administrative authority over underlying infrastructure (the administrative domain) used to implement resources. The administrator uses management systems to configure and manage resources within the administrative domain.

2. Infrastructure Service User: The entity which accesses an infrastructure service in order to obtain, examine, modify and destroy resources.

3. Infrastructure Service Provider: The entity which offers an infrastructure service that may be used by an *infrastructure service user* to obtain, examine, modify and destroy resources.

7.4.1 Administrator

This section specifies the responsibilities of the administrator that are relevant for the security aspects of cloud networking:

1. The administrator must ensure that the auditing mechanism has been invoked as per its desired frequency.

2. The administrator shall display the current security status of the underlying resources (or an overall security status) to the user. The user might want to learn about the security status of a larger set of resources than regularly displayed, and this might be dependent upon the information sharing policy of the administrator.

3. The administrator must ensure that all the security parameters are implemented as specified by the SLA and any discrepancies must be reported to the user.

4. The user must be made aware of the different security policies which might be selected through her SLA, and the performance-cost-security trade-offs for each: For example, a user might be impressed with the reduced cost benefit of utilizing data centres in countries outside Europe, but she must also be informed about the apparent reduction in overall security and increased risks associated with the storage.

5. The administrator shall ensure smooth delegation and cooperation of different entities, while implementing the security parameters in cases where the resources might not be in its direct control.

6. The administrator shall ensure that the security parameters are mapped in the best possible way on the diverse resource set. Therefore, the security parameters should be mapped on any underlying technology (for example, OpenFlow or L2/L3 VPN) without any observable difference in the security characteristics.

7.4.2 Infrastructure Service User

This section lists the primary responsibility of the Infrastructure Service user:

1. The Infrastructure Service user must be aware of the performance-cost-security trade-offs for each possible security policy chosen by her. Once she signs the SLA, she is accepting the risks associated with each option.

The Infrastructure Service User has the following rights:

1. Depending upon the information sharing policy of the individual administrators, the user has the complete right to view the status and capabilities of the underlying resources.

2. The user is free to choose any security policy, unless it interferes with the federal policy of the area (where either the resources or the administrator is situated - we might extend this to cover the area in transit as well) or with the resource management policy of the administrative authorities.

The responsibilities of the Infrastructure Service provider shall be covered in Section 7.5.

A detailed description of the different roles, and their interactions with the architecture, is covered in Section 2

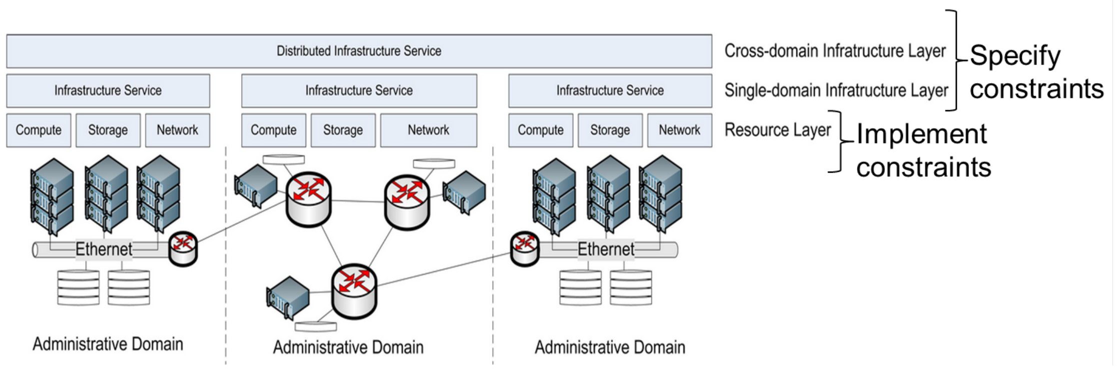

Figure 7.3: Three layer model goals

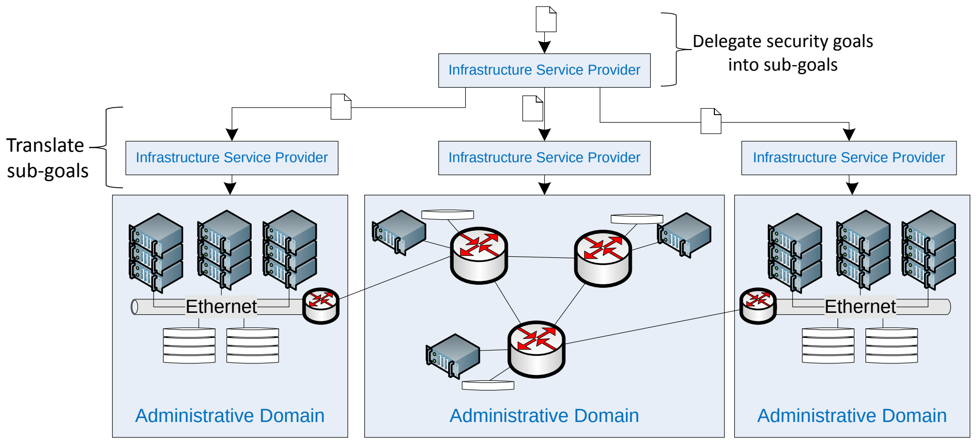
Infrastructure Service Provider
Delegate security goals into sub-goals
Translate sub-goals
Infrastructure Service Provider
Infrastructure Service Provider
Infrastructure Service Provider
Ethernet
Ethernet
Administrative Domain
Administrative Domain
Administrative Domain

Figure 7.4: Delegation of security goals

7.5 Relation to Technical Architecture

The security framework involves the security goal translation function as its backbone, which shall accept service requests, monitoring results and topology information, and generate constraints on parameters, which are characterized with respect to the underlying resources.

As is evident from Figure 7.3, the security goal translation function shall be used to specify the parameter constraints at the architecture layers. Thus, the goals that have been specified at the distributed control plane through the Infrastructure Service user SLAs will be translated into concrete constraints, at the architecture layers. These constraints would then be realized by the administrator. As covered earlier, all the goals might not map fairly directly to the parameter constraints, and thus additional goals (or extensions on the security parameters) might be defined at the individual architecture layers to formulate specifications that are relevant for constraining the resources controlled at that layer.

The detail layer at which the parameter constraints will be defined and realized at each of the architecture layers (cross-domain and single-domain) is still open and shall be explored during the prototyping phase.

In Figure 7.4, a new role is introduced, termed as the "Infrastructure Service provider". The Infrastructure Service provider shall offer an infrastructure service that may be used by an *infrastructure service user* to obtain, examine, modify and destroy resources. The working of the

Infrastructure Service provider, both for the single-domain and the cross-domain infrastructure scenario, has been explained in detail in Section 2, through the Figures 2.2 and 2.3. For the security goal translation, the Infrastructure Service provider will be responsible for translating the goals into sub-goals and/or security parameters, which then need to be delegated, either directly in the case of single-domain architecture, or indirectly through an additional Infrastructure Service provider, in the case of a cross-domain architecture, to the respective administrators in control of the actual resources.

These sub-goals/security parameters will then be translated into parameter constraints (with the help of some additional goals/parameters defined to allow a seamless translation). These constraints will then be applied on the respective resources by the resource management function.

Finally, the interfaces and communication mechanism between the individual administrators, to allow information exchange and ease goal translation, shall be similar to the interfaces and mechanisms described by the multi-domain infrastructure layer investigation team. It won't make sense to define new interfaces and communication mechanisms to allow information exchange, unless it poses a security threat and violates the proposed security policy. This shall be explored further in the prototyping stage as well.

7.6 Security Goal Translation Example

As covered in the description for the overall goal translation function, the function shall be agnostic about the hierarchies of the control domains, and hence will behave in the same manner at any layer of the service hierarchy. It will accept a security control goal (also termed as a security objective) from the owner/issuer of the goal at a higher layer in the control plane hierarchy, and shall translate it into sub-goals (which are further propagated to the lower layers in the control hierarchy), or parameters which need to be further constrained with respect to specific resources. The goal translation function shall accept 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). Figure 7.5 shows the hierarchical arrangement of the goal translation functions for security goals.

The following is a high-level example of the security goal translation function's working. The workings already described as a part of the overall goal translation function shall not be repeated.

A security goal G for obtaining a connection between points A and B could comprise of the following terms:

1. Ensure that the resources/administration authorities are placed only in the European economic area.
2. Ensure that the encryption algorithm used to maintain confidentiality uses at least 2048 bit keys (or 224 bits for Elliptic Curve Cryptography (ECC)).
3. The network operator(s) involved are running ISO 27005 (Information security risk management) certified equipment.

The process of the security goal translation could then comprise of the following steps:

1. The goal translator must identify a set C of plausible FNS compositions that provide the connectivity between points A and B.
2. The goal translation function then identifies the administrative domains for each composition X in the set C, and generates the connectivity-specific sub goals to be propagated to each composition's administrative domains. This procedure is described in detail during the description of the overall goal translation function, and hence hasn't been repeated.

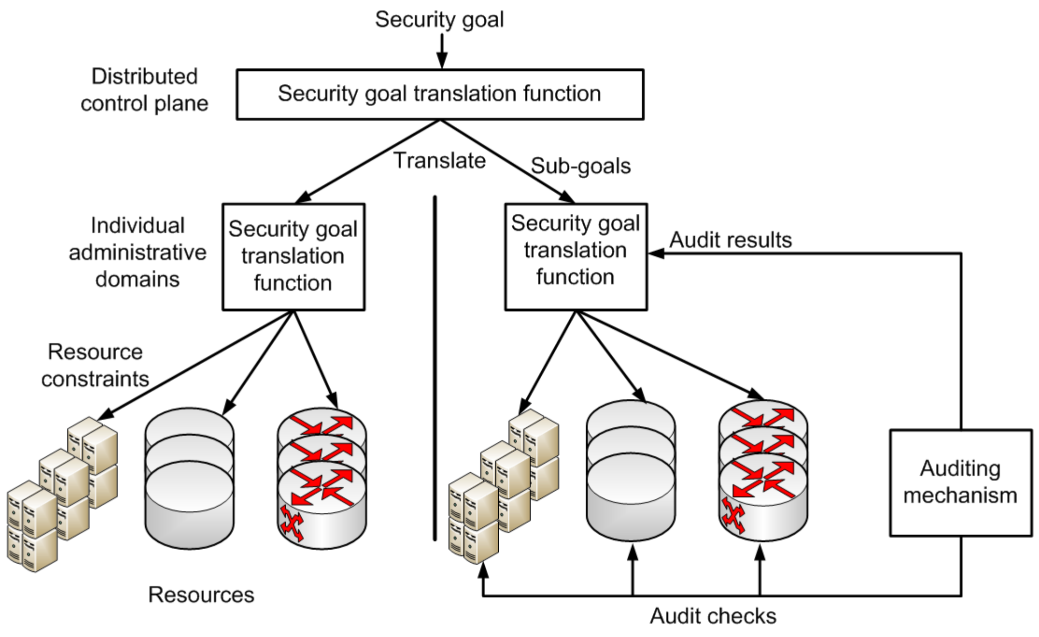
Security goal
Distributed control plane
Security goal translation function
Translate
Sub-goals
Individual administrative domains
Security goal translation function
Security goal translation function
Audit results
Resource constraints
Auditing mechanism
Resources
Audit checks

Figure 7.5: Hierarchical arrangement of goal translation

3. The security sub-goals will be simultaneously generated along with the general sub-goals, and will require the interaction of additional modules for the fulfilment of the same.

4. The fulfilment of the specification that 'resources/administration authorities are placed only in the EEA' can either be fulfilled by sub-goals that check the actual geographic location of the resources/authorities through the use of a geo-positioning software, or sub-goals that include the use of trusted platform modules in the hardware to maintain trust relations and detect the geographic locations with a very high success rate.

5. The key length check involves a sub-goal which invokes a trusted piece of software code to verify the actual key length (and its entropy during generation). The level of trust that need to be maintained between the different resources and the administrative authorities/service providers/users is currently hazy and need to be discussed further.

6. Similarly, the key length checks can also involve testing the Public Key Infrastructure (PKI) for its efficacy and efficiency. This would involve a sub-goal to invoke an independent auditing scheme to verify the working and security levels of the underlying PKI.

7. The ISO 27005 certification of the underlying resources has to be separately carried out by an independent third-party accredited with carrying out certifications. For verifying that the resources are indeed certified with ISO 27005, trusted mechanisms may be developed which check whether the resources have the certification or not.

8. Finally, an access control policy model has to be integrated into the security goal translation function's primitives, such that it can translate the goals/sub-goals specified by the higher control plane into valid access control rules to be enforced on the underlying resources.

7.7 Access Control Policy

Implementing a fine-grained access control policy successfully is the backbone of a large number of security services. The model chosen to define the policy must not only define all the possible access control rules that might be desired, but should also be easy to learn and understand, well documented and easily implementable.

A major drawback of the discretionary access control model [57] is that addition of new subjects (active entity), actions or objects (passive entity) results in a total update of the overall security policy. The policy is defined as being composed of ternary expressions, whereby each expression shall define the subject, its object and the possible actions permissible on them. On the other hand, the role based access control model [58] introduces the concept of session and role hierarchy. The major drawback to the model is that the role hierarchy doesn't usually map directly to the organization hierarchy, which increases the overall complexity of implementing the model in the organization.

Therefore, a model needs to be chosen which would solve the above disadvantages and allows context and organization specific rule formulation. For example, we might have a rule which is only possible under a specific context, for example allow modification of the code repository by the project lead, if technical member == project lead. Moreover, we require organization specific roles, for example allow modification of the code repository by the organization, if technical member == organization. Finally, a model has to be chosen on the ease of mapping between the language used to describe the overall security goals and the language used to model the access control policies in the chosen model. Easy inter-language mapping should also be complemented by the ease of mapping between the language of the security model and the underlying resources.

7.8 Conclusion and Further Steps for Security

The security chapter has covered an overview of the overall security methodology, how to obtain and translate security goals using the security goal translation function and how it integrates with the existing high-level of the architecture. A small description of security parameters has also been included, along with the security specific roles and responsibilities of the involved entities.

Future steps would include the selection and placement of an auditing mechanism to ensure that the security management function is behaving as per the specified guidelines and no discrepancies slip under the radar. An auditing mechanism is extremely important for the security aspects of the architecture, as many issues might not be detected by the fault management function.

Moreover, the access control policy is an indispensable part of managing the security of any architecture. Therefore, the selection of a relevant access control policy model with the desirable features covered above is of the utmost importance. Ideally, the model shall promote scalability and complexity of the infrastructure, and should be easy to integrate with the heterogeneous infrastructure utilized by cloud networking.

Furthermore, security functionality needs to be included into the architecture that allows to isolate the FNS, e.g., to enforce security zones.

It is planned to integrate some of the security functionalities into the prototype, e.g., the access control functionality.

8 Relation to Use Cases

A number of use cases for cloud networking were reported in deliverable D.A.1 [3] with further development for supporting business cases. These use cases describe the type of services the CloNe architecture is expected to support.

The use cases are grouped under two broad scenarios: elastic video distribution, including live video distribution, video on demand, video conferencing, and distributed gaming; and dynamic enterprise, based around enterprise services deployed on cloud infrastructure including business goal management, travelling worker support (remote auditing use case), virtual desktop, and external collaboration (media production use case).

The use cases are used to guide the further development of the architecture. Here we take one representative use case for each scenario, video on demand (from the elastic video distribution scenario) and media production (from the dynamic enterprise scenario) to demonstrate two applications of the architecture. The first scenario emphasizes use of distributed, in-network processing nodes, whereas the later emphasizes flexible connectivity and provisioning of network and cloud resources.

8.1 Video on Demand Use Case

The video on demand service involves delivering video content to consumers on request. The service is represented in the Figure 8.1.

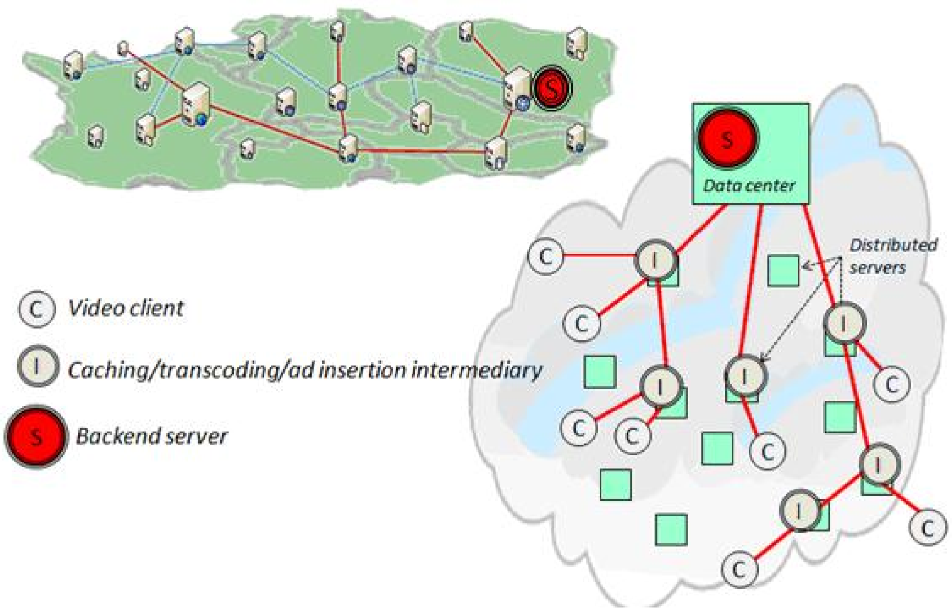

Figure 8.1: Video on Demand Use Case

The consumers are represented by video clients connected to a network provider. The source video content is managed and distributed from a central data centre by a service provider. There may be one or more network providers between the source data centre and the clients. The network providers and data centre are infrastructure service providers in our architecture.

The scalability of the service and optimal use of the network is achieved by placing caching and transcoding servers across the networks and managing connectivity between these servers, the source data centre and the clients. As the demand for content changes, the clients change, and distribution of clients change the optimal placement of cache/transcode servers and connectivity will change. The CloNe architecture supports the use case by providing a means for the service provider to dynamically distribute the cache/transcode servers across the network providers and manage the network usage. The video service provider will interact with the CloNe system through the Infrastructure Service Interface. It will describe the virtual infrastructure required to implement the service in a high level language, relating the number and placement of servers to the number and clustering of clients. It may describe the network connectivity among servers and to the clients in terms of metrics such as closeness and bandwidth/latency requirements.

DCP will decompose the virtual infrastructure into fragments to be delivered to infrastructure service providers who will map the fragments to their own infrastructure, placing and deploying virtual resources according to their knowledge of their physical infrastructure. DCP may interpret metrics such as closeness and bandwidth in terms of linkage among infrastructure service providers whereas individual infrastructure service providers will interpret closeness and bandwidth in terms of their physical network topology. Hence the high level requirements need not be phrased in terms low level topology, respecting the network providers desire not to expose information about their physical infrastructure. The infrastructure service providers will deploy the cache/transcode servers using their internal compute resource interfaces and may assign storage using their internal storage resource interfaces. Network connectivity will be established using the internal network resource interfaces. Where network resources are required to connect across different infrastructure service providers, DCP will provide information describing how this is to be achieved (i.e. provider gateway/remote resource identification). This may involve interaction between network resource management systems at the resource layer to establish resource connection.

We aim to address the resource management challenge from the perspective of Video on Demand (VoD) use case. The VoD service involves delivering video content to consumers on request. We aim to come up with statistical methods in order to gain insight on the potential future behaviour of the system under consideration and increase its elasticity and flexibility at runtime. In order to test the statistical methods we modelled and simulated user behaviours with an objective to reproduce a realistic and versatile workload and the corresponding data set. Lack of good quality video usage databases is also another factor that motivated us to artificially generate it. We adopt epidemic models for this purpose. Figure 8.2 representing user behaviour (workload in terms of % of viewers) in a typical case involving 5 different videos introduced in the server. However, this model need to be investigated further to make it more realistic and hence more useful.

Our final objective is to embed the user behaviour generating model in the Grid 5000 test-bed to generate actual video traces and user requests. With the input from this model in our algorithm (statistical method) we would then have some perception about the probable behaviour of the system, leading the service to be reactive to the change (sudden) in user behaviour.

8.2 Enterprise Use Case

Enterprises are traditionally a crucial sector for IT and network services, therefore this should represent one of the most promising targets for cloud services in the future. Enterprise services are characterized by strict requirements in terms of parameters such as reliability, availability, performance and security. Lack of guarantees to fulfil these parameters has been perceived as a major obstacle against the widespread adoption of cloud services by enterprises. Hopefully this

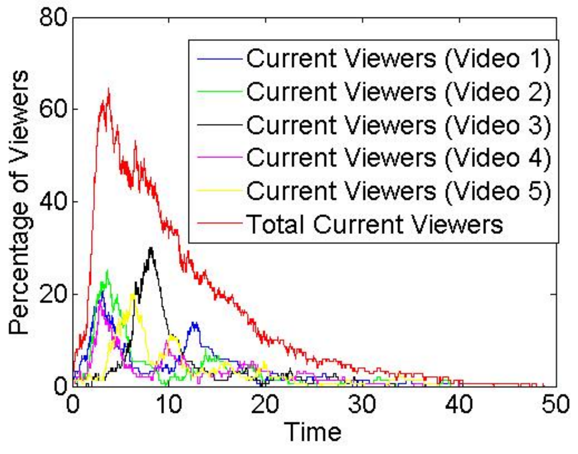

Figure 8.2: Video on Demand Use with an example of the evolution of aggregate viewers for five different videos introduced in the server at the same time

limitation will be tackled in the next generation of cloud services. Cloud networking plays a crucial role in this scenario. Cloud networking provides enterprises with the capability to add dynamic flexible network provisioning to cloud computing and flexibly scale services to build on demand and pay-per-use IT/IS solutions. With FNS capabilities, enterprises 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 [Ref. Del D-A.1].

The media production use case is a practical realization of the enterprise scenario. As described in deliverable D-A.1, this use case involves a TV channel based in UK using cloud networking for its IT services to extend its own production facilities for use by sub-contractors, which in this case is a video animation company located in Japan. In order to achieve SLAs at minimum cost and maximize resource efficiency, the cloud resources should be located near the sub-contractor premises, in Japan. The CloNe architecture supports the use case by providing the means to dynamically establish secure networking connectivity between the cloud site located in Japan, the animation company and the channel production systems. The TV channel plays the service customer role and the animation company acts as business partner. The TV channel is a customer of a local cloud service provider, which, for reasons of resource use efficiency, delegates the provision of the service to a cloud service provider located in Japan. Based on this use case, it is possible to highlight a number of requirements to be accommodated by the architecture, in particular by DCP:

- Security: the cloud may hold highly confidential and proprietary enterprise information. Among other things, this means that DCP should place constraints on which network resources are eligible to be involved in a flash network slice. Section 6 provides information on this particular topic.

- Isolation: the companies involved in the use case, should be fully isolated from each other, apart from the specific resources to be shared. This implies that the network is able to guarantee the isolation of resources, while at the same time enabling collaboration and sharing environment among the involved partners. Network technologies widely used in enterprise such as Network-based VPNs have a strong reputation for isolation among different private networks sharing the same infrastructure. Newer technologies such as network virtualisation can also offer good isolation properties.

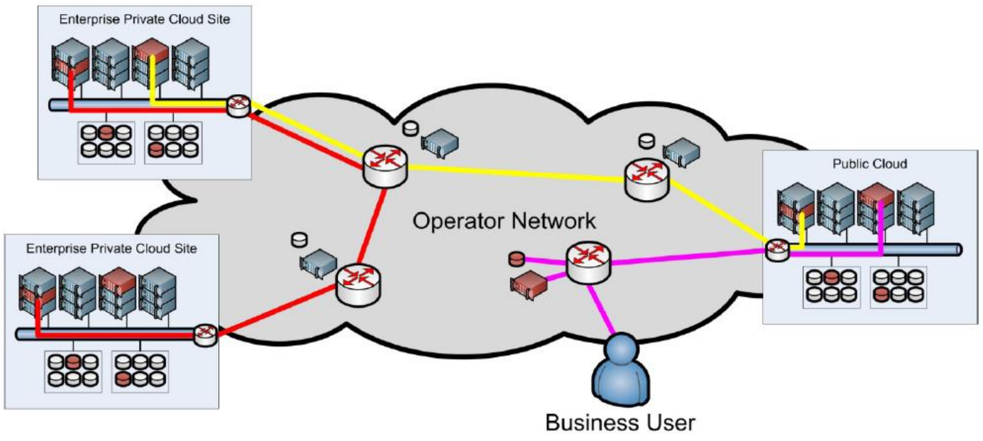

Figure 8.3: Enterprise Use Case

- Technology heterogeneity: the service is expected to involve network domains with very heterogeneous characteristics. Therefore, the architecture is expected to decouple the service from the specific characteristics of the underlying infrastructure, by providing an abstraction layer through implementation of goal translation functions. In the CloNe architecture it is up to the Infrastructure Provider to apply whatever mapping rules are appropriate taking into account the specificities of the network technology and the service goals.

- Inter-Domain: as the use case shows, in many cases a flash network slice is likely to cross multiple network domains, which raises several inter-domain networking issues. DCP should be able to construct a coherent service based on network resources that belong to multiple administrative domains run by independent providers. The solution depends on the specific network technology.

- Dynamic provisioning and reconfiguration: the use case requires the provisioning of the service on a fully on-demand automated basis. This requires the network establishment and reconfigurations to be controlled by the user with minimal human intervention (ideally, no intervention at all) from the involved service provider(s). In addition, live service migration should be supported whenever necessary, while generating minimum service disruption. Dynamic network control is enabled by network virtualisation. For older network virtualisation forms (e.g. VPNs) this represents a challenge, which has to tackled by the CloNe architecture.

- Enterprise-grade service robustness: as mentioned before, CloNe must provide enterprise-grade reliability, availability, performance and security. This requires the utilization of network technologies that comply with such requirements, which is the case with mature technologies such as L2/L3 VPNs. Newer technologies, like Openflow or full-blown network virtualisation cannot offer as yet the same level of robustness, but the utilization of such technologies in enterprise environments should not be ruled out.

9 Related Work

The distributed nature of a control plane for future networks along with the necessity of the real time and more dynamic nature of the services has made the task of defining architecture very challenging. There are several other projects going on at the moment and the following relevant projects are highlighted as of interest.

1. ETICS (Economies and Technologies for Inter-carrier Services) [59] is an EU FP7 project, that aims to create an interconnection between different network service providers keeping quality of service in mind. ETICS is prototyping a control plane implementation as well as assessing performance to demonstrate how their architecture might improve the services as well reduce operating expenditure and complexity. This project is particularly interesting for SAIL as it addresses the issue of multiple telecommunication operator interaction.

2. GEYSERS (Generalised Architecture for Dynamic Infrastructure Services) [60] is an EU FP7 project that aims at defining an architecture for managing and controlling virtual infrastructure integrating network and IT resources. GEYSERS is designing a control plane architecture for the optical networks by expanding existing standard solutions (GMPLS and PCE). It aims to efficiently integrate (optical) network services and IT Services.

3. The Network Service Interface Working Group (NSI-WG) is also dealing with the on-demand provisioning of ”end to end circuits” [61]. The Inter-Domain Controller Protocol (IDCP) that they have implemented, is currently used for dynamic provisioning of network resources for different vendors. A dedicated working group on the control plane itself (NMWG-CP) also exists in NSI.

4. RESERVOIR was a EU FP7 funded project aimed at building a cloud federation of similar (homogeneous) data centres connected via the best-effort Internet.

5. CloudAudit [62] provides a common interface and name space that allows cloud computing providers to automate the Audit, Assertion, Assessment, and Assurance (A6) of their infrastructure (IaaS), platform (PaaS), and application (SaaS) environments and allow authorized consumers of their services to do likewise via an open, extensible and secure interface and methodology.

6. OrBAC [63] was developed inside the RNRT MP6 project (communication and information system models and security policies of healthcare and social matters). The purpose of this project is to define a conceptual and industrial framework to meet the needs of information security and sensitive healthcare communications.

7. Cloud controls matrix (CCM) [64] is specifically designed to provide fundamental security principles to guide cloud vendors and to assist prospective cloud customers in assessing the overall security risk of a cloud provider.

ETICS defines an overall end-to-end SLA lifecycle taking into account of business criteria. This research project depicts a seven layer end-to-end service-specific SLA lifecycle method namely creation, SLA and trust certificate publications, negotiation, validation, provisioning and invocation, monitoring and termination. In addition research on network capability requirements for important network features such as inter-carrier/domain path computation, admission control for session services, congestion notification, resilience and security are being conducted. In case of inter-carrier network scenarios, as the ETICS architecture describes, the neighbouring service providers along the data path have also to trigger admission control procedures for the resources on the corresponding inter-carrier connections (for example point-to-point links or peering points) in order to assure that the composed end-to-end service will get the desired network resources along the whole data path. In order to carry out admission control in the context of e.g. session initiation or connectivity setup, it would be necessary that the admission control process gets access to the SLA descriptions and thus to the specified SLA parameters. ETICS is prototyping a control plane implementation as well as assessing performance to demonstrate how their architecture might improve the services as well reduce operating expenditure and complexity. ETICS provisions Quality of Experience (QoE) for the end user by provisioning end-to-end admission and congestion control.

In spite of its advanced research in the area of inter-carrier services, ETICS does not consider network provisioning of cloud data centre networks. This project is particularly interesting for SAIL as it addresses the issue of multiple telecommunication operator interaction.

GEYSERS extends the Virtual Infrastructure concept to the optical layer. A VI is defined as a collection of virtualised optical network and IT resources with coordinated management and control processes. The fact of defining a VI over a set of virtual resources provides the possibility of sharing the underlying physical infrastructure among different operators (virtual resource market place), and granting them isolation. Dynamic VI provisioning mechanisms are integrated into the infrastructure definition, producing the potentiality to modify the VI capabilities in order to align them with the VI usage needs at any given instant.

Optical network is one of the key components, which consists of various optical devices such as Optical Cross Connects (OXC), Reconfigurable Optical Add-Drop Multiplexers (ROADM) or Optical Transport Network (OTN) resources. Optical resources are featured by the enormous bandwidth of each wavelength, and in particular, the emerging flexible grids technology enables the more efficient utilization of optical spectrum and flexible bandwidth allocation. IT resources comprise another important category of managed objects in GEYSERS (remote visualization, computing and storage). In order to build flexible VIs, these resources are abstracted and partitioned into Virtual Resources (VRs), which are attached to the VI and exposed to the Virtual Infrastructure Operator (VIO). Each VR contains a well-defined, isolated subset of capabilities of a given physical device. Taking the example of an optical device, the VRs are built on its total amount of ports or its wavelengths, and control switching exclusively between these allocated ports and wavelengths.

GEYSERS explore a disruptive approach for on-demand high capacity Net+IT environment provisioning. GEYSERS extends the IaaS approach to the Network. GEYSERS model can be used to solve Cloud Networking issues but is not focusing, as SAIL, on the specific networking issues of current Clouds. These two projects are complementary as they explore the same paradigm (virtual infrastructures) at different levels of the protocol stack and focus on different use cases.

RESERVOIR depicted a layered architecture that included three levels: service [10], virtual execution environment (data centre) [6] and resources (where the network bit belongs into [65]). The service manager layer was kept unaware of the federation work to provide the illusion of virtually infinite resources underneath. The service manager was the sole entry point to the system, in charge of uniquely naming and identifying the resources, enforcing service scalability and user-level SLAs. After optimizing users' requests for some performance and cost goals, it passed its VM/Storage/data centre network request to data centre layer in charge of deploying it. When a request to a data centre was locally irresolvable, it was forwarded to another federated data centre. This federation level just implied a forwarding mechanism of deployment requests and monitoring information across relay nodes in every data centre.

Since a service could contain VMs being deployed in separate data centres, some network level federation was also required. This was done by including an overlay isolating user networks from each other (in the data centre, overlay connecting data centre hypervisors) and connecting them across data centres over the Internet by means of a gateway, a forwarder node that sent packets from a data centre to the other across the Internet.

In spite of its advanced features, RESERVOIR did not consider that the network can host compute and storage resources and how this fact can affect the performance of the application thus deployed. Also, RESERVOIR did not contemplate the fact that connectivity is often not enough for the needs of some applications (e.g. real time, low latency, big data and the like). The Quality of Experience (QoE) for the end user is typically dependent on network parameters (e.g. latency), which can only be achieved by exploring how to implement a tighter interaction at several points:

1. between the data centre and the network operators, right at the border with the network operator to set a series of technologies for connectivity, select among the most convenient one and let data centre and network edge nodes negotiate the best terms for establishing the required connectivity with the needed QoS to enable real QoE.

2. between network operators themselves to ensure the required QoS terms that the best-effort Internet has no means of guaranteeing (e.g. establishing a temporary peering agreement to deliver traffic in a faster manner to another edge of that network in return for some fee.

Although, as stated in [65], cloud providers cannot be expected to coordinate their network maintenance, network topologies, and more so with one another, adding dynamism in the way data centres are communicated among each other is an essential requisite for many of today's more demanding and profitable applications (e.g Netflix is already consuming most of the bandwidth in the U.S.). In this regard, SAIL can be considered as one of RESERVOIR's heirs, dealing with the integration (federation) of heterogeneous data centres across (even more) heterogeneous network operators. These network operators can also host computational or storage resources and negotiate the connectivity terms from the edges of two or more data centres to help achieving application's QoE needs.

The corresponding working group to NSI-WG in OGF have suggested a hierarchical approach in terms of the label switching paths (LSP) for a MPLS-TP Control plane reference model [66]. These hierarchical LSPs or "H-LSPs" as they call them provide scalability, facilitates operations administration maintenance (OAM) and maintenance end point (MEP) within each provider as well as across inter provider network to network interface. These MEPs collect network performance information and also have some other functionality relevant to SAIL's interest. Both ETICS and GEYSER (and NSI-WG to some extent) deal with the optical networks which are an important basis of the future Internet and thus relevant for SAIL. An issue for SAIL is which control architecture to adopt: hierarchical or peering. As has been described in this document we have chosen to adopt the hierarchical model as the primary approach, as peering can be implemented at a level within this model. In terms of network services and infrastructure as a service framework the hierarchical control plane provides more flexibility in terms of configuration, accessibility and availability and at the same time this structure facilitates the separation of the functional aspects of converged services (from service consumer to physical networking and IT infrastructure). This approach is also being implemented in the GEYSER project and has been adopted in the HIPerNet [24] software for the purpose. The context of mentioning the hierarchical approach as well as HIPerNet is important, since they are proven answers to future Internet definition; but the choice between the peering and hierarchical approach is still open to be debated.

The layered model adopted in the CloNe architecture is a common theme in several of these projects. Some, such as GEYSERS, include higher service levels and make network control a central layer of the architecture. CloNe takes a slightly different approach with the notion of a virtual infrastructure composed of interconnected virtual resources of different types including network, compute, and storage, as the central theme, hinting towards a different decomposition and delegation of control of infrastructure across administrative domains. A feature of CloNe is the development of a new network resource, the flash network slice, and its role within infrastructure as a service. SAIL follows the experience of the 4WARD project [67] which developed architecture for development of interoperable networks of differing technologies. As part of this work 4WARD designed and implemented a layered approach to full network virtualisation. In SAIL CloNe takes a fresh view with a focus on cloud computing and providing a virtual network resource type for use in this context. Flash network slices will be used within virtual infrastructures and to connect virtual infrastructures to each other and to end users. CloNe will focus on rapid programmatic control (create, manage and destroy) of flash network slices as a component of a virtual infrastructure driven by high level description.

The concept of in-network management introduced in 4WARD will to a large extent constitute the foundation on which CloNe management processes will be further developed for addressing the aforementioned challenges in the cloud. The cloud network management will follow the principles of decentralized operation, addressing the challenges of scalability, adaptability, control, reliability and resource usage efficiency. Whereas only basic concepts of business goal management were provided in 4WARD, goal translation will in SAIL further address the challenges of simplified controllability and interoperability for expressing and controlling business goals and high-level objectives. Adaptive fault management will in SAIL be focused on fault detection and handling based on the concepts of in-network management, but will address problems both in individual virtual layers and across several layers, enabling effective and preventive fault management of resources in the cloud. Tools and approaches for resource management will be further extended and developed in SAIL, supporting self-management processes, scalability, adaptivity and controllability in the cloud while enabling efficient usage of resources in the cloud.

CloudAudit (codename: A6) provides a common interface and name space that allows cloud computing providers to automate the Audit, Assertion, Assessment, and Assurance (A6) of their infrastructure (IaaS), platform (PaaS), and application (SaaS) environments and allows authorized consumers of their services to do likewise via an open, extensible and secure interface and methodology. The benefits to the Cloud Service Provider are to enable the automation of typically one-off labour-intensive, repetitive and costly auditing, assurance and compliance functions and provide a controlled set of interfaces to allow for assessments by consumers of their services. The benefits to the "consumer" of the Cloud services or their duly-authorized representatives are to provide a consistent and standardized interface to the information produced by the service provider.

CloudAudit integration shall provide technology agnostic representative schema and data structures mapped to existing compliance, security and assurance frameworks, which can be integrated into the security framework planned for CloNe architecture.

OrBAC provides a well defined access control policy model, which can be integrated into the security framework for CloNe, and shall enable fine-grained access control of the resources. The OrBAC API has been created to help software developers introduce security mechanisms into their software. This API implements the OrBAC model, which is used to specify security policies and also implements the AdOrBAC model [68], which is used to manage the administration of the security policies. The MotOrBAC tool [69] has been developed using this API to edit and manage OrBAC security policies. OrBAC has only been realized on homogeneous systems (such as firewalls) or at the software level. Current work will include the implementation of the OrBAC formalism on the underlying heterogeneous hardware environment.

The CSA (Cloud Security Alliance) CCM (Cloud Controls Matrix) provides a controls framework that gives detailed understanding of security concepts and principles that are aligned to the Cloud

Security Alliance guidance in 13 domains. The foundations of the Cloud Security Alliance Controls Matrix rest on its customized relationship to other industry-accepted security standards, regulations, and controls frameworks such as the ISO 27001/27002, ISACA COBIT, PCI, and NIST, and will augment or provide internal control direction for SAS 70 attestations provided by cloud providers. This controls framework will serve as the backbone for evaluating the security levels of the CloNe architecture.

10 Conclusions and Future Work

The architecture presented in this document is under continuing development using an iterative approach. The main focus of the first year of the CloNe work package has been to gain an understanding of cloud networking. The basis of this understanding is the concept of a network resource called the flash network slice and how it relates to the cloud computing infrastructure as a service 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 cloud networking architecture proposed in this document goes beyond the state of the art in the area of infrastructure service cloud. The flash network slices provide 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 flash network slices provide dynamic connectivity services with the level of performance and reliability expected by enterprise applications to be deployed in the cloud. The proposed architecture allows users to specify measurable performance goals associated to resources allocated in the infrastructure. Yet another new concept proposed by cloud networking is the deployment of computing and storage resources within network. These resources will allow a finer level of distribution of service than the one provided by existing data centres.

10.1 Future Work

We expect to further refine the CloNe architecture as a result of lessons learned from the development of the prototype. Moreover, additions to the architecture will be made to cover for extra aspects that were identified during the first 12 months of the project. Those include the need for user authentication solution across cloud providers; provisioning of scalable inter-provider flash network slices; a model for delegation across business boundaries or subcontracting; and inter-provider resource management.

The proof-of-concept prototype based on the architecture presented in this document is currently under development. The key aspects of the architecture will be demonstrated in the prototype, namely: cross-administrative allocation of resources; dynamic scaling of flash network slices; allocation of computing resources in a distributed cloud; amongst others. An experimental infrastructure is being built that spans Ericsson's site in Sweden, HP's site in UK, and Institute Telecom's site in France. This experimental infrastructure will include real cloud infrastructures implemented at the three sites, connected by an emulated L3VPN network. These will be extended with implementations of the flash network slices and management systems to implement a distributed cloud networking infrastructure service supporting multiple applications. Example applications based on the CloNe use cases will be implemented to test the implementation. The use-cases chosen for implementation were the elastic video distribution and media production.

We also expect to further explore the relationship between CloNe and the other technical work packages of SAIL: NetInf and OConS. NetInf is constructing object addressable networks as a service overlay network. CloNe has the potential to support spontaneous deployment and optimisation of NetInf server nodes, managing the connectivity among nodes, and supporting dynamic geographical placement and reconfiguration decisions. OConS is developing highly adaptive network connectivity services supporting multipath and multipoint connections. While CloNe implements flash network slices based on OpenFlow, VPNs or virtualised networks, multipath services proposed by OConS could as well be used in OConS enabled networks.

A Management Algorithm Results

A.1 Goal Translation and Monitoring of High-Level Objectives

The rest of this subsection gives an account for our approach to goal translation and the restrictions on high-level goals. Let ISP be an infrastructure service provider that manages several resources $r_1, \ldots, r_{m_r}$. Assume for simplicity that they are of the same type and that they expose identical low-level parameters $\pi_1, \ldots, \pi_{m_\pi}$ used for configuring the resource. For each $i = 1, \ldots, m_r$ and $j = 1, \ldots, m_\pi$, we associate to the parameter π_j a family of distributions $\Delta_{i,j}(x)$, where x ranges over the values of π_j, obtained from monitoring of measured characteristics of the resource r_i. Note that π_j is not required to be an actual configurable parameter of the resource: it suffices that the resource manager can configure the resource on the basis of the value of π_j.

Assume that ISP offers a service[1] S with service parameters $s_1, \ldots, s_{m_s}$. Note that the service parameters can be both high and low-level parameters. We say that a *goal for S* is a set of constraints on the service parameters $s_1, \ldots, s_{m_s}$. It is up to ISP to specify exactly which parameters should be included among the service parameters $s_1, \ldots, s_{m_s}$ for S. This decision depends on several factors, some of which we call *market*, *abstraction*, and *measurement factors*. The market and abstraction factors decide which parameters the infrastructure service provider would like to include in the goal specification language, while the measurement factor restricts which parameters that may be used in specifying goals.

The market factor reflects the (estimated, measured, or otherwise perceived) customer requirements on the service S. If for example S is a connectivity service, then the service parameters may (most likely) include parameters for the specification of the endpoints of the connection, start and end times for the connection, and possibly for the locations of the nodes over which the connection is routed. The abstraction factor reflects characterisations of services that cannot be (or are not conveniently) controlled directly on resource level. This includes QoS parameters such as bandwidth, jitter, drop rate, and delay. It may further include security specifications and fault management instructions. For example, one may want to specify that a service respects some privacy policies.

The measurement factor determines whether the infrastructure service provider can translate a goal expressed on a service performance parameter. In order to be able to translate a goal on a service parameter s_k, we need a function g that tells how the performance parameters $\pi_{i,1}, \ldots, \pi_{i,m_\pi}$, where i corresponds to the resource r_i with $i = 1, \ldots, m_r$, should be set for maximising the probability of obtaining the specified constraint on s_k. One approach to defining this function is based on monitoring and measurements of the parameters $\pi_{i,1}, \ldots, \pi_{i,m_\pi}$, $i = 1, \ldots, m_r$, and s_k. The idea is to find a functional dependence f of s_k on $\pi_{i,1}, \ldots, \pi_{i,m_\pi}$, and to define the function g in terms of the inverse of f. (Note that the inverse of f may not exist as a function, though, in which case g becomes indeterministic.) Based on these measurements, it is in some cases possible to obtain the function g from (a subset of) the set of distributions over the values of s_k into the set of values of $\pi_{i,1}, \ldots, \pi_{i,m_\pi}$, $i = 1, \ldots, m_r$ so that, for some distribution $D(s_k)$, if the resource is configured according to $\pi_{i,1}, \ldots, \pi_{i,m_\pi}$, then s_k will take values in accordance with $D(s_k)$. The possibility of finding such a (partial) function depends on the relation between the parameters $\pi_{i,1}, \ldots, \pi_{i,m_\pi}$ and s_k (i.e., the function f). If there is a linear dependence, then it will be relatively easy to find the

[1] In general, an infrastructure service provider could offer more than one service.

function g. Most likely, the relation will not be simple, and we may have to use techniques from machine learning to learn or estimate the function f relating $\pi_{i,1}, \dots, \pi_{i,m_\pi}$ to s_k, $i = 1, \dots, m_r$. Whenever the function g can be defined (for some distribution D over s_k), we say that a goal expressed in terms of the parameter s_k, can be be translated by the goal translation function and that s_k is an *admissible* high-level parameter. The set of goals that may be expressed for a particular infrastructure service provider will be restricted to the goals containing only constraints over admissible parameters.

We shall now give a sketch of goal translation. Assume that ISP, the infrastructure service provider as specified above, is requested by an actor A to fulfil the goal $G = \{D(s_k)\}$, where $D(s_k)$ is a distribution over the values of the service parameter s_k for the offered service S. The distribution is a constraint on s_k. (In general a goal will consist of constraints on several service parameters). We assume for this exposition that the service S requires exactly one of the managed resources, and thus that there is no need to decompose the goal into several subgoals. The translation of G proceeds in several steps.

The current status of the resources is first retrieved in terms of what values $v_i = (v_{i,1}, \dots, v_{i,m_\pi})$ of the parameters $\pi_i = (\pi_{i,1}, \dots, \pi_{i,m_\pi})$ the resource r_i can offer, $i = 1, \dots, m_r$. In general, the values in v_i may be scalar, intervals, or probability distributions. For each $i = 1, \dots, m_r$, based on v_i and the gathered statistics via the measurements of π_i and s_k, we can obtain a distributions $\delta_{i,k}$ over the values of s_k measured with respect to resource r_i with parameter settings according to v_i.

For each $i = 1, \dots, m_r$, the distribution $\delta_{i,k}$ is compared with the objective $D(s_k)$ for s_k, to see how well $\delta_{i,k}$ matches $D(s_k)$. The measure of match between $\delta_{i,k}$ and $D(s_k)$ is obtained via a statistical measure which is left out in this exposition (see [70]). Through an optimisation criterion, the goal translation function selects the resource r_j for which $\delta_{j,k}$ is considered to be the best match to $D(s_k)$. In order to find configuration objectives for the resource r_j, we make a second optimisation among the values of $\pi_{j,\ell}$ induced by $v_{j,\ell}$, $\ell = 1, \dots, m_\pi$ for finding the values $o_j = (o_{j,1}, \dots, o_{j,m_\pi})$ that maximise the probability of r_j obtaining the distribution $\delta_{j,k}$ over s_k.

Based on the selection of r_j the goal translation function can then offer a service implemented by r_j to the requester A with the promise to fulfil the goal G with a certain probability p. The probability p reflects the discrepancy of match between $\delta_{j,k}$ and $D(s_k)$; the likelihood of $\delta_{j,k}$ given o_j; potential uncertainties in the retrieved resource status information v_j; as well as, in applicable cases, the probability of $o_{j,\ell}$ being drawn from the distribution $v_{j,\ell}$, $\ell = 1, \dots, m_\pi$. If the offered service is accepted by the requester, then the resource management receives $o_{j,1}, \dots, o_{j,m_\pi}$ as its configuration objectives for resource r_j.

While the service S, implemented using r_j, is running, the parameters $\pi_{j,\ell}$ are continuously monitored via the associated distributions $\Delta_{j,\ell}(o_{j,\ell})$, $\ell = 1, \dots, m_\pi$. If the monitored values start to deviate from the distributions $\Delta_{j,\ell}(o_{j,\ell})$, and if the resource and fault management functions are unable to correct the behaviour locally, then the goal translation function will redo the translation process attempting to find other solutions (possibly involving replacing the resource). If the goal translation cannot find a replacement solution, then it will report to the requester A that the goal cannot be met according the agreed performance characteristics. From that point either a re-negotiation may take place, or the requester may decide to use an alternative infrastructure service provider for its services.

A.2 Fault Detection in Virtualised Systems

As an example, we have tested the scalability of the spatio-temporal event correlation protocol [26] for an increasing network size in both the number of layers and the number of nodes, using synthetically generated topologies. The topologies were generated using the the Erdös-Rényi method [71]. For an increasing number of virtual nodes N, a series of experiments was performed for $N = \{500, \ldots, 5000\}$ and a fixed number of layers $L = 3$. For an increasing number of layers L, the series of experiments was performed for $L = \{3, \ldots, 18\}$ and a fixed number of nodes $N = 3000$. For each layer n the number of nodes N_n decreased by a fraction of 0.7.

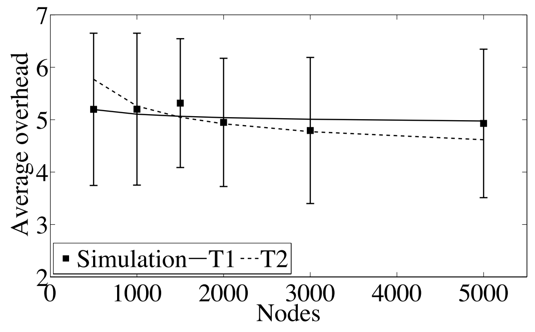

Figure A.1: Average overhead for an increasing number of nodes, with theoretical path lengths calculated from the entire graph (T1) and as a sum of individual layers (T2).

We observe that the overhead changes in accordance with the theoretical average path length between any pair of nodes (Figure A.1), which for an ER graph is known to be proportional to $l_{rand} \propto \frac{ln(N)}{ln(pN)}$, where N is the number of nodes and p the connection probability [71]. We here compare the simulation results with two aspects of the theoretical path length, based on calculations from the whole graph and as a sum of individual layers. As the theoretical average path length does not take into account multiple layers, the theoretical lines have been shifted to simplify comparison (based on minimising the sum of differences towards zero for each theoretical line and the data points).

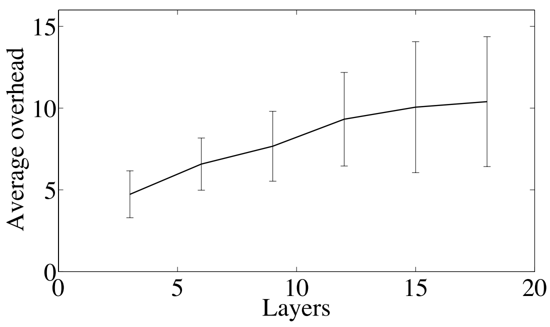

Figure A.2: Average overhead for an increasing number of layers.

In Figure A.2, we see that the overhead increases with the number of layers. Naturally, the overhead is increased as the payload have to travel a greater distance to the root cause layer when the number of layers is large. Since we in this case generated graphs with fewer nodes for each layer, the overhead eventually levels out as the proportion of small layers grows.

A.3 Predicting and Modelling Virtual Infrastructures: initial results

Information dissemination in a social system (Gossip or Buzz) can be viewed as an epidemic spreading in a population network. We studied relevant epidemic models and then drew analogy between epidemic outbreaks and user behaviour patterns in a video on demand system. Spreading of an epidemic in a population can be categorised into a few characteristics which is related to the infection under consideration. We elaborate on this model in this Appendix section. We consider two main parameters, which are β (rate of contamination) and γ (rate of recovery) [72].

In this model there can be two events that change the state variables N_s, N_i, N_r [73].

Event 1 One extra infection (N_s decreases) with rate $\beta.N_s.(N_i + N_r)/N$

Event 2 One extra infection (N_r increases) with rate $\gamma.N_i$

This epidemic model can be adopted to represent the way information spreads among the users in a video on demand system. For a Video on Demand (VoD) system, susceptible S means potential viewers of a video, while contagious C refers to people who already watched (or are currently watching) the video and who can spread the information about it. However, in order to get the number of current viewers of a video, which directly defines the workload on the system, we divide the contagious compartment into two subsets. One, that we note I, contains the current $N_I(t)$ viewers of a video, and the complementary set called R, comprises the $N_R(t)$ persons who already watched the video. N_s, N_i and N_r are functions of time. β is the transmission rate at which information spreads from the current or past viewers to the potential viewers. Finally, to complete our model, we consider that the watch time of a video is exponentially distributed with mean γ^{-1}. In our model $N_s + N_i + N_r = N$ (fixed). This workload generator represents a **Markov chain** [74].

Figures A.3 and A.4 detail some preliminary results that we obtained from our model. We ran Matlab simulations for single video and multiple videos with β changing from 1 to 1.5 to 2 and $\gamma = 0.25$. The first figure simulates a scenario for a single video with threshold effect that includes the importance of video popularity in the dynamics of the model. We see that the rate of change of current viewers gets higher after it crosses a defined threshold (we consider the first threshold to be one fourth and the second one to be one half of the total population without losing generality, we are investigating different possibilities of defining the threshold in order to make it more realistic). The second figure simulates a scenario for five different videos, that have been introduced at the same time in the server and have the same values of β and γ. This scenario is more realistic, because in a video on demand system users have freedom of choosing different videos (one at a time). We see that most of the time (except video 3 and video 4) different videos reach their maxima at different times. It is a general case of video on demand system where it is not surprising that one video becomes more popular than the others and after everybody has watched it, another video gets its turn and drives the system. Thus the process continues.

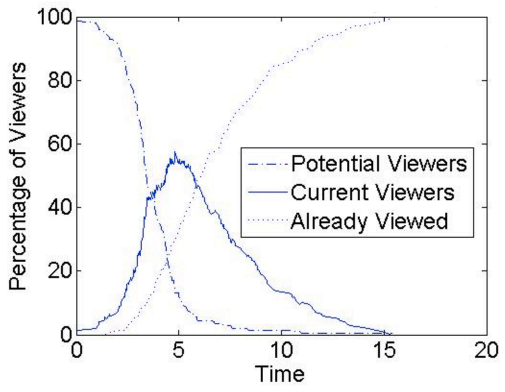

Figure A.3: An example of the evolution of viewers with changing value of β (Popularity) for one video.

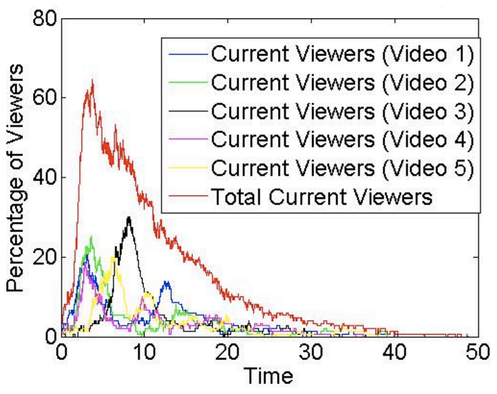

Figure A.4: An example of the evolution of viewers with changing value of β (Popularity) for the aggregate of five different videos.

A.4 Mapping of Virtual Networks with Cloud Resources

Here is presented some preliminary work on an algorithm for mapping cloud networking requests (where network resources are mapped into VNs) based on node stress and link bandwidth. In a first step, a pre-selection of possible physical hosts will be done, taking into account the user's access point (i.e. the geographical place where the user will access the service), the network conditions and service network requirements. The mechanism will determine the possible solutions, i.e., one or several physical hosts able to allocate the cloud resources which, at the same time, can have a network path (between the physical host and the user's access point) able to fulfil the required QoS to access the host.

[48] focuses on the problem of virtual network resources embedding, taking into account the heterogeneity of the physical network. For this problem, it is proposed an efficient heuristic that solves the resource allocation problem. The proposed algorithm is inspired by the concepts of node and link stress, defined in [43]. We are extending this algorithm by integrating the mapping of network resources (routers and links) along with the placement of cloud resources in the network. Two algorithms, which differ in the server stress formula, are presented and a comparative analysis between both is done. Note that these formulas are applied to individual servers scattered through the network and not to data centres. The mapping of the request is done by combining the methodology used in [48] with the present server stress formulas. For the moment the algorithm does not take latency into account, but as a fundamental parameter in the future the algorithm will consider it.

$$Ss = \frac{NumberofActiveVMs}{(N.CPU - Load) + \sigma} \frac{FreeRAM}{FreeRAM - ReqRAM + \sigma} \frac{FreeHDD}{FreeHDD - ReqHDD + \sigma} \tag{A.1}$$

$$Ss = \frac{NumberofActiveVMs}{FreeRAM(N.CPU - Load)FreeHDD + \sigma} \tag{A.2}$$

Formula A.1 considers the amount of RAM and HDD associated to the VM that we want to map. Thus, in this situation the stress of the server will depend on the VM that is trying to be mapped, while in Formula A.2 that is not taken into account.

In the simulations both physical substrate and VNs had 20% of the nodes as servers (rounded to the higher integer) and the remaining 80% as routing nodes. Each scenario was done for 12 runs, each with 500 time units. The same subtract and VN requests were used for the study of both algorithms, with a VN request tax of $\lambda = 2$ VNs per time unit, with an average duration of

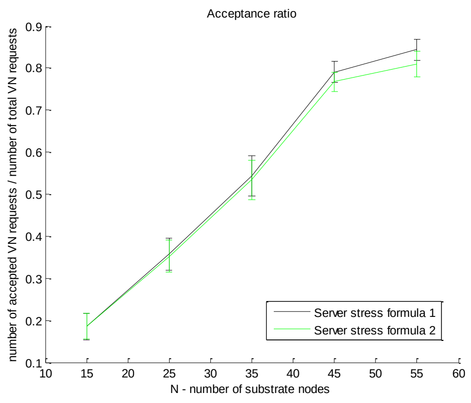

Figure A.5: Acceptance rate (number of accepted requests / number of total requests) varying the number of physical subtract nodes.

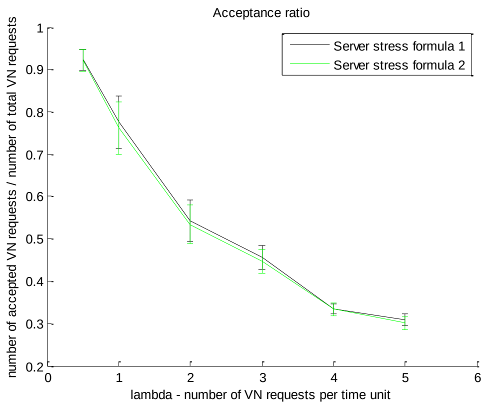

Figure A.6: Acceptance rate (number of accepted requests / number of total requests) varying the number of requests per time unit.

$1/\mu = 20$ time units (Poisson arrivals and exponentially distributed duration). The virtual servers' characteristics are based in the Amazons offerings. The results presented have a 90% confidence interval.

From Figures A.5 and A.6 it is possible to observe that Formula A.1 presents significantly better results than Formula A.2 especially when the number of subtract nodes increases. This leads us to conclude that it is better to perform the mapping considering the characteristics of the virtual servers along with those from the physical ones. The difference is not very sharp since the algorithms only differ in the server's stress formula and are the same for the routing nodes stress formula. It is noted that the acceptance rate rises as the number of substrate nodes increases since there is more capacity and more nodes where the mapping can be done. This increase is sharper when the number of substrate nodes is lower since in these conditions the acceptance rate is limited by low number of nodes of the substrate. As for the acceptance rate behaviour according to the variation of the number of requests per time unit it naturally decreases as the number of requests increases. Even though in this situation it is possible to see a slightly better performance from Formula A.1.

Bibliography

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

[2] 4WARD Consortium. Virtualisation approach: Concept. Technical report, ICT-4WARD project, Deliverable D3.1.1, Sep. 2009.

[3] Tapio Levä, Joao Gonçalves, Ricardo Jorge Ferreira, and Johan Myrberger. D-2.1 (D-A.1) Description of project wide scenarios and use cases. Technical report, FP7-ICT-2009-5-257448-SAIL/D.A.1 Deliverable, Feb 2011.

[4] Benoit Tremblay et. al. D-2.2 (D-A.2) Draft Architecture Guidelines and Principles. Technical report, FP7-ICT-2009-5-257448-SAIL/D.A.2 Deliverable, July 2011.

[5] libvirt, the virtualization api. http://libvirt.org.

[6] Borja Sotomayor, Ruben S. Montero, Ignacio M. Llorente, and Ian Foster. Virtual infrastructure management in private and hybrid clouds. *IEEE Internet Computing*, 13:14–22, 2009.

[7] Opennebula. http://www.opennebula.org/.

[8] 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.

[9] H. Medhioub, I. Houidi, W. Louati, and D. Zeghlache. Design, implementation and evaluation of virtual resource description and clustering framework. In *Advanced Information Networking and Applications (AINA), 2011 IEEE International Conference on*, pages 83 –89, march 2011.

[10] 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.

[11] J. Gottlieb, A. Greenberg, J. Rexford, and J. Wang. Automated provisioning of bgp customers. *Network, IEEE*, 17(6):44 – 55, nov.-dec. 2003.

[12] Xu Chen, Z. Morley Mao, and Jacobus Van der Merwe. Pacman: a platform for automated and controlled network operations and configuration management. In *Proceedings of the 5th international conference on Emerging networking experiments and technologies*, CoNEXT '09, pages 277–288, New York, NY, USA, 2009. ACM.

[13] William Enck, Thomas Moyer, Patrick McDaniel, Subhabrata Sen, Panagiotis Sebos, Sylke Spoerel, Albert Greenberg, Yu-Wei Eric Sung, Sanjay Rao, and William Aiello. Configuration management at massive scale: system design and experience. *IEEE J.Sel. A. Commun.*, 27:323–335, April 2009.

[14] Don Caldwell, Anna Gilbert, Joel Gottlieb, Albert Greenberg, Gisli Hjalmtysson, and Jennifer Rexford. The cutting edge of ip router configuration. *SIGCOMM Comput. Commun. Rev.*, 34:21–26, January 2004.

[15] Nick Feamster and Hari Balakrishnan. Detecting bgp configuration faults with static analysis. In *Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation - Volume 2*, NSDI'05, pages 43–56, Berkeley, CA, USA, 2005. USENIX Association.

[16] Yi Wang, Eric Keller, Brian Biskeborn, Jacobus van der Merwe, and Jennifer Rexford. Virtual routers on the move: live router migration as a network-management primitive. *SIGCOMM Comput. Commun. Rev.*, 38:231–242, August 2008.

[17] N.M. Mosharaf Kabir Chowdhury and Raouf Boutaba. A survey of network virtualization. *Comput. Netw.*, 54:862–876, April 2010.

[18] Eric Keller and Jennifer Rexford. The "platform as a service" model for networking. In *Proceedings of the 2010 internet network management conference on Research on enterprise networking*, INM/WREN'10, pages 4–4, Berkeley, CA, USA, 2010. USENIX Association.

[19] Martín Casado, Teemu Koponen, Rajiv Ramanathan, and Scott Shenker. Virtualizing the network forwarding plane. In *Proceedings of the Workshop on Programmable Routers for Extensible Services of Tomorrow*, PRESTO '10, pages 8:1–8:6, New York, NY, USA, 2010. ACM.

[20] Openstack cloud software. http://www.openstack.org/.

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

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

[23] Guilherme Koslovski, Sebastien Soudan, Paulo Gonçalves, and Pascale Vicat-Blanc. Locating Virtual Infrastructures: Users and InP Perspectives. In *12th IEEE/IFIP International Symposium on Integrated Network Management - Special Track on Management of Cloud Services and Infrastructures (IM 2011 - STMCSI)*, Dublin, Ireland, May 2011. IEEE.

[24] Fabienne Anhalt, Guilherme Koslovski, and Pascale Vicat-Blanc Primet. Specifying and provisioning Virtual Infrastructures with HIPerNET. *ACM International Journal of Network Management (IJNM) - special issue on Network Virtualization and its Management*, 2010.

[25] Tram Truong Huu, Guilherme Koslovski, Fabienne Anhalt, Pascale Vicat-Blanc Primet, and Johan Montagnat. Joint elastic cloud and virtual network framework for application performance optimization and cost reduction. *Journal of Grid Computing (JoGC)*, 2010.

[26] R. Steinert, S. Gestrelius, and D. Gillblad. A Distributed Spatio-Temporal Event Correlation Protocol for Multi-Layer Virtual Networks. In *GLOBECOM 2011 (to appear)*, 2011.

[27] R. Steinert and D. Gillblad. Towards Distributed and Adaptive Detection and Localisation of Network Faults. In *2010 Sixth Advanced International Conference on Telecommunications*, pages 384–389. IEEE, 2010.

[28] R. Steinert and D. Gillblad. Long-Term Adaptation and Distributed Detection of Local Network Changes. In *IEEE Global Telecomm. Conf. GLOBECOM*, pages 1–5, 2010.

[29] Rerngvit Yanggratoke, Fetahi Wuhib, and Rolf Stadler. Gossip-based resource allocation for green computing in large clouds (long version). Technical Report TRITA-EE 2011:036, KTH Royal Institute of Technology, `https://eeweb01.ee.kth.se/upload/publications/reports/2011/TRITA-EE_2011_036.pdf`, April 2011.

[30] F. Wuhib, R. Stadler, and M. Spreitzer. Gossip-based resource management for cloud environments. In *CNSM 2010*, pages 1–8, October 2010.

[31] M. Isard. Autopilot: automatic data center management. In *SIGOPS Oper. Syst. Rev., vol. 41, no. 2*, pages 60–67, 2007.

[32] A. Verma, P. Ahuja, and A. Neogi. pmapper: power and migration cost aware application placement in virtualized systems. *Middleware 2008*, pages 243–264, 2008.

[33] C. Clark, K. Fraser, S. Hand, J.G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield. Live migration of virtual machines. In *Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation-Volume 2*, pages 273–286. USENIX Association, 2005.

[34] M.R. Hines and K. Gopalan. Post-copy based live virtual machine migration using adaptive pre-paging and dynamic self-ballooning. In *In Proceedings of the 2009 ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*, pages 51–60, New York, NY, USA, 2009.

[35] J. Nogueira, M. Melo, J. Carapinha, and S. Sargento. A distributed approach for virtual network discovery. In *GLOBECOM Workshops (GC Wkshps), 2010 IEEE*, pages 277–282, 2010.

[36] Lucian Luciu et. al. Deliverable D.C.1 - Architectural Concepts of Connectivity Services. Technical report.

[37] Timothy Wood, Alexandre Gerber, K. K. Ramakrishnan, Prashant Shenoy, and Jacobus Van der Merwe. The case for enterprise-ready virtual private clouds. In *Proceedings of the 2009 conference on Hot topics in cloud computing*, HotCloud'09, Berkeley, CA, USA, 2009. USENIX Association.

[38] A. Farrel, J.P. Vasseur, and J. Ash. A path computation element (pce)-based architecture. IETF RFC4655, August 2006.

[39] E. Oki, T. Takeda, JL. Le Roux, and A. Farrel. Framework for pce-based inter-layer mpls and gmpls traffic engineering. IETF RFC5623, September 2009.

[40] N. Mosharaf K. Chowdhury and Raouf Boutaba. Network virtualization: State of the art and research challenges. *IEEE Communications Magazine*, 47(7):20–26, July 2009.

[41] Norbert Egi, Adam Greenhalgh, Mark Handley, Mickael Hoerdt, Felipe Huici, Laurent Mathy, and Panagiotis Papadimitriou. A platform for high performance and flexible virtual routers on commodity hardware. *SIGCOMM Comput. Commun. Rev.*, 40(1):127–128, 2010.

[42] M. Siraj Rathore, Markus Hidell, , and Peter Sjödin. Data plane optimization in open virtual routers. In *Proceedings of IFIP Networking 2011*, Valencia, Spain, May 2011.

[43] 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.

[44] Jens Lischka and Holger Karl. A virtual network mapping algorithm based on subgraph isomorphism detection. In *VISA '09: Proceedings of the 1st ACM workshop on Virtualized infrastructure systems and architectures*, pages 81–88, New York, NY, USA, 2009. ACM.

[45] 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):1729, 2008.

[46] 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.

[47] N. Farooq Butt, M. Chowdhury, and R. Boutaba. Topology-awareness and reoptimization mechanism for virtual network embedding. *NETWORKING 2010*, pages 27–39, 2010.

[48] 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.

[49] I. Houidi, W. Louati, and D. Zeghlache. A distributed virtual network mapping algorithm. In *Communications, 2008. ICC '08. IEEE International Conference on*, pages 5634 –5640, 19-23 2008.

[50] Z.B. Houidi and M. Meulle. A new vpn routing approach for large scale networks. In *Network Protocols (ICNP), 2010 18th IEEE International Conference on*, pages 124 –133, oct. 2010.

[51] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. Openflow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review*, 38(2):69–74, 2008.

[52] Virtual private network consortium. http://www.vpnc.org/.

[53] Openflow switch specification - version 1.1.0. http://www.openflow.org/documents/openflow-spec-v1.1.0.pdf, feb 2010.

[54] Open networking foundation. http://www.opennetworkingfoundation.org/.

[55] Virtual networks research group - irtf. http://http://irtf.org/vnrg/.

[56] Focus group on future network itu-t. http://www.itu.int/en/ITU-T/focusgroups/fn/Pages/Default.aspx/.

[57] Michael A. Harrison, Walter L. Ruzzo, and Jeffrey D. Ullman. Protection in operating systems. *Commun. ACM*, 19:461–471, August 1976.

[58] Serban I. Gavrila and John F. Barkley. Formal specification for role based access control user/role and role/role relationship management. In *Proceedings of the third ACM workshop on Role-based access control*, RBAC '98, pages 81–90, New York, NY, USA, 1998. ACM.

[59] ETICS: Economics and Technologies for Inter-Carrier Services. https://www.ict-etics.eu/overview.html.

[60] GEYSERS: Generalised Architecture for Dynamic Infrastructure Services. http://www.geysers.eu/index.php/theproject/goals.

[61] Inter-Domain Controller (IDC) Protocol Specification, Open Grid Forum Network Service Interface (NSI) Working Group. http://www.controlplane.net/idcp-v1.1-ogf/draft-gwdi-nsi-idcp-2010-sep-01.pdf, September 2010.

[62] CloudAudit, June 2011. http://cloudaudit.org/.

[63] A. Abou El Kalam, R. El Baida, P. Balbiani, S. Benferhat, F. Cuppens, Y. Deswarte, A. Miège, C. Saurel, and G. Trouessin. Organization Based Access Control. In *4th IEEE International Workshop on Policies for Distributed Systems and Networks (Policy'03)*, June 2003.

[64] CSA CCM Leadership Team. Cloud security alliance cloud controls matrix v1.1. Technical report, Cloud Security Alliance, 2010.

[65] Benny Rochwerger, David Breitgand, Amir Epstein, David Hadas, Irit Loy, Kenneth Nagin, Johan Tordsson, Carmelo Ragusa, Massimo Villari, Stuart Clayman, Eliezer Levy, Alessandro Maraschini, Philippe Massonet, Henar Munoz, and Giovanni Toffetti. Reservoir - when one cloud is not enough. *Computer*, 44:44–51, 2011.

[66] MPLS-TP Control Plane Framework, IETF Internal Draft. http://tools.ietf.org/html/draft-ietf-ccamp-mpls-tp-cp-framework-05, January 2011.

[67] 4WARD. http://www.4ward-project.eu/index.php.

[68] Frédéric Cuppens and Alexandre Miège. Administration Model for Or-BAC. In *Computer Systems Science and Engineering (CSSE'04)*, volume 19, May 2004.

[69] Fredéric Cuppens, Nora Cuppens-Boulahia, and Céline Coma. MotOrBAC : un outil d'administration et de simulation de politiques de sécurité. In *Security in Network Architectures (SAR) and Security of Information Systems (SSI), First Joint Conference*, June 6-9 2006.

[70] B. Bjurling, R. Steinert, and D. Gillblad. Translation of probabilistic qos in hierarchical and decentralized settings. In *The 13th APNOMS 2011 (to appear)*, 2011.

[71] Réka Albert and Albert-László Barabási. Statistical mechanics of complex networks. *Rev. Mod. Phys.*, 74(1):47–97, Jan 2002.

[72] Alain Barrat, Marc Barthelemy, and Alessandro Vespignani. *Dynamical processes on complex networks*. Cambridge University Press, 2008.

[73] Linda J. Allen. *An Introduction to Stochastic Processes with Biology Applications*. Prentice Hall, April 2003.

[74] E. Seneta. Markov and the birth of chain dependence theory. *International Statistical Review*, 64(3):255–263, 1996.