Objective FP7-ICT-2009-5-257448/D-2.3

Future Networks

Project 257448

"SAIL – Scalable and Adaptable Internet Solutions"

D-2.3
(D.A.3) Final Harmonised SAIL Architecture

Date of preparation: **2013-02-28**
Start date of Project: **2010-08-01**
Project Coordinator: **Thomas Edwall**
Ericsson AB

Revision: **1.0**
Duration: **2013-02-28**

Document Properties

Document Number:	D-2.3
Document Title:	**(D.A.3) Final Harmonised SAIL Architecture**
Document Responsible:	Benoit Tremblay (EAB)
Document Editor:	Benoit Tremblay (EAB)
Authors:	Pedro A. Aranda (TID), Holger Karl (UPB), Matthias Keller (UPB), Dirk Kutscher (NEC), Paul Murray (HP), Manuel Peuster(UPB), Christoph Robbert(UPB), Fabian Schneider (NEC), Peter Schoo (FHG), Lucian Suciu (FT), Asanga Udugama(UHB)
Target Dissemination Level:	PU
Status of the Document:	Initial version
Version:	1.0

Production Properties:

Reviewers:	Holger Karl (UPB), Paul Murray(HP)

Document History:

Revision	Date	Issued by	Description
1.0	2013-02-28	Benoit Tremblay	Initial revision

Disclaimer:

This document has been produced in the context of the SAIL Project. The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7/2010–2013) under grant agreement n° 257448.

Abstract:

This document presents the overall SAIL architecture for the future internet. The SAIL architecture is build on three pillars NetInf, CloNe and OConS. While these components are described in more details in specific documents (*D.B.3 Final NetInf Architecture* [1], *D.D.3 Refined CloNe Architecture* [2] and *D.C.2 Architectures and mechanisms for connectivity services* [3] respectively, the present document aspires at presenting a global view, putting those pieces together in a global perspective and identifying synergies emerging from that integration.

Keywords:

SAIL, architecture, information-centric network, open connectivity services, cloud networking, future internet

Executive Summary

This document is a public deliverable of the Scalable Adaptive Internet Solutions (SAIL) EU-FP7 project [4] and presents the overall technical architecture developed during the project.

We first reassess our decision of selecting an architectural approach focusing on the integration the building blocks rather than targeting a monolithic architecture.

After presenting an overview of the architecture and identifying the interfaces, we describe for each pair of work packages overlap, complementarity and interactions.

For NetInf and CloNe, we illustrate how NetInf can be deployed over CloNe using the elastic feature available in CloNe. To achieve that, some clarifications to the CloNe architecture were required. To that purpose, a new component Application Deployment Toolkit (ADT) is introduced and described.

For NetInf-OConS integration, choices need to be made where to put routing and forwarding decisions. OConS can provide a good support for those functions at the packet level but NetInf does this using Named Data Object (NDO) information. This can be solved either by providing network information to the NetInf layer or to provide some information on object aggregation to the convergence layer. We use multi-path/multi-source forwarding as an example to illustrate those two possible solutions.

CloNe can use OConS as an improved packet transport. In addition, the OConS services are better used for establishing and monitoring the Flash Network Slice (FNS) in the OConS domain. To realise this, a mapping is required between the CloNe Infrastructure Service Interface (ISI) (DCP, OCNI) and the OSAP provided by OConS.

For each of the three SAIL cross-work package Themes (Security, Network management and Interprovider), we summarise the work that has been done during the SAIL project.

We finally identify some open issues and elements for further study in the conclusion.

Contents

1 Introduction

In Scalable Adaptive Internet Solutions (SAIL), we envision that the "Ingredients for the Network of the Future will be Content, Connectivity, and Cloud" [5]. These three essential ingredients are served by the three main components of the SAIL solution. Network of Information (NetInf) provides a way to retrieve content without a priori knowing its location. Open Connectivity Services (OConS) provides advanced networking mechanisms and orchestrate them to improve the overall efficiency of the network. Cloud Networking (CloNe) integrates cloud and network allowing to distribute the cloud further in the network as well as strengthening the connectivity between the different components of the cloud applications through network virtualisation and domain federation.

In the previous deliverables (D.B.3, D.C.2 and D.D.3 [1–3]), we have focused on describing the architecture of these individual components. In the current document, we will describe how these different solutions integrate and interact with each other. These interactions were initially identified in D.A.2 [6] and then refined in each of the components' architecture description. Experimentation and prototyping activities then allowed their validation and clarification (see D.A.9 [7] for a description of these activities). We assume the reader to be relatively familiar with those documents as they are often referred to in the present text.

Before describing each of the internal system interactions, we first motivate and reassess in Chapter 2 our conscious decision of not proposing a monolithic architecture for the future internet but rather to propose a more loose architecture where the main components interact but can be deployed independently over time or space. In the same chapter, we reiterate the overall SAIL architecture which was presented in D.A.2 [6]. Then, for each pair of components from NetInf, OCons and CloNe, we examine pair-wise the interactions, complementarity and potential overlap. First, in Chapter 3, we present how NetInf can benefit from the elasticity services available in CloNe, describing in detail the CloNe component managing elasticity. In Chapter 4, we present the interactions between the NetInf convergence layer and the OConS orchestration and exemplify that through the use of multi-path mechanism from OConS. In Chapter 5, we present how OConS and CloNe collaborate in establishing and monitoring Flash Network Slice (FNS). In Chapter 6, we present how selected overall Themes were addressed throughout the different components identifying synergies and commonalities when applicable. Finally, we present a summary and conclude this architecture document.

2 Overall Architecture

The intent of this chapter is to describe

- potential architecture approaches (Section 2.1),
- our perspective on an overall architectural integration (Section 2.2),
- an overview of the SAIL architecture and the interactions between individual work package architectures (Section 2.3).
- the timescales at which components work (Section 2.4).

2.1 Monotlithic Architectures vs. Interacting Building Blocks

The goal of a tightly integrated, hermetic, possibly even monolithic architecture for the entire technical scope of a research project is alluring. There are evident benefits:

- Required functionality can be split at the best possible points (best separation of concerns, least information exchange, best information hiding),
- individual functions can be harmonised with each other,
- duplication of functionality can be reliably avoided,
- optimisation can happen on a firm basis.

On the other hand, such a broad-sweeping architecture holds a number of challenges as well:

- It can be very difficult to deploy since an all-or-nothing buy-in is required.
- Given its sheer complexity, it is extremely hard to develop and reach the best possible goal.
- Users are forced to use functionality they might have no interest in.
- Legacy systems can be impossible to integrate into/with a monolithic approach.

These challenges can be better addressed via smaller building blocks that are geared towards a specific goal, that do a small job reasonably well. Those blocks can interact both with other blocks of the given architecture and with other, legacy systems. While clearly an integrated architecture is superior in potential, in practice we believe that this potential can be neither achieved in research nor realised in real life. Moreover, this approach makes it much easier to experiment with and use multiple different solutions for one problem – instead of believing in and hoping for "one true way", we do embrace diversity not only for research, but also for real-world deployment (akin to the UNIX philosophies by Raymond, Gabriel and others).

Nevertheless, a set of interacting building blocks does face downsides and disadvantages. Common risks are

- Function duplication
- Reduced optimisation potential
- More complex interfaces (owing to suboptimal functional separation, the need to carry more-than-required information across an interface)

2.2 SAIL Approach

To determine which approach suits better for SAIL, let's have a look at our vision, revise the objectives and identify the applicable constraints imposed to the project.

The SAIL vision is based on three elements: Content, Connectivity and Cloud. For each of those elements, SAIL has selected and developed specific technical solutions: NetInf, OConS and CloNe. While NetInf, OConS and CloNe share some functions, the set of functions required to implement those technical elements do not fully overlap and in some cases do necessitate different approaches.

SAIL targeted a smooth migration from today's internet to the network of the future with a start of deployment in three to five years. Moreover, the maturity of the technical solutions developed in SAIL is not homogeneous, cloud being already deployed while Information Centric Networking (ICN) still has some technical and business challenges to be solved before hitting the road. Also, each element of the solution may target different players in different industries (e.g. content distributors, network operators, data centre providers, ...). This mandates partial and progressive deployment of the SAIL solutions. It also calls for support for the co-existence of legacy technologies for those elements of the solution that will be first deployed until the full SAIL solution is adopted and deployed.

The ambitious goals targeted by the project advocated for early and parallel prototyping of components of the technical solutions. This can hardly be done if we first need to elaborate a complete and consistent architecture that takes into consideration all aspects of the problem that we are tackling.

Looking at all the inherent or self-imposed constraints and objectives, we are hence thoroughly convinced that a monolithic approach to the architecture makes no sense in SAIL. This is why, after more attention and reflection, we have decided to pursue the integration of building blocks approach that we selected at the beginning of the project.

We shall discuss in the following chapters where we believe that the risks of the selected architecture approach materialised in SAIL but first we present an overview of the main architecture components and interfaces in the section below.

2.3 Architecture Overview

The SAIL core components NetInf, OConS and CloNe have been presented numerous times in the different SAIL deliverables and publications. In this section we identify where those components interface and interact while the details of those interfaces and interactions are provided in the following chapters. Figure 2.1 overviews the SAIL architecture and the major relations between the three components.

The interfaces identified in the figure can be classified in three groups: interfaces towards applications provided by SAIL components, interfaces towards external functions used by SAIL components and internal interfaces allowing SAIL components to interact.

One assumption that we have made is that the maturity of these components will vary and the components may be deployed at different points in time. Although the SAIL components must take advantage of each other when deployed in the same context they must be able to be deployed without the others. This motivates to maintain the external interfaces from NetInf and CloNe towards the legacy connectivity functions (IP and L2 networks). The deployment and migration strategy for the SAIL components is described in a separate deliverable (D.A.4 [8]).

The external interfaces both towards applications and external functions are already described in the respective component architecture deliverables.

The focus of the coming chapters is to examine in more details the internal interactions.

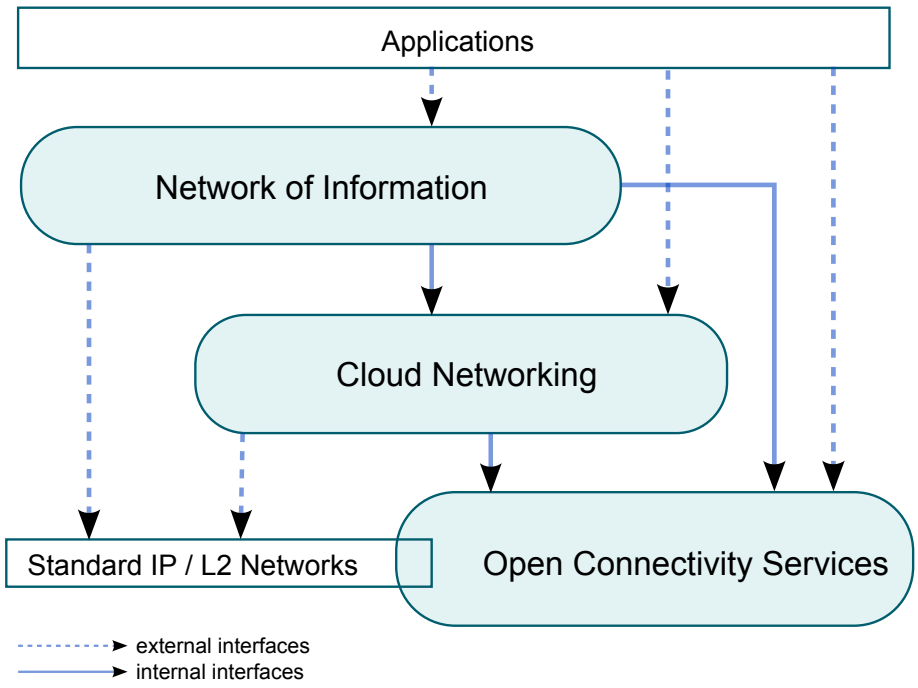


Figure 2.1: SAIL Architecture Overview

2.4 Timescales

It bears mentioning that the three main components not only complement each other functionally, they also work at different time scales (see Figure 2.2).

- CloNe works at the longest timescales: deploying, moving, copying, booting new virtual machines, setting up networking between a new virtual machine and its peers, and initializing it properly is a process that takes place on the order of at least tens of seconds, typically minutes, sometimes even hours. It is a mechanism that can react to smoothed out traffic and load changes and that can provision additional resources for longer trends. It is well suited to pre-planned deployment as well.
- OConS has two different aspects:
  - The actual packet forwarding in an OConS context obviously has to happen at line speed.
  - The orchestration process – recognizing opportunities for treating packets or flows differently as well as establishing the necessary actions – takes place on the order of at least several packets, typically on the order of tens to hundreds of milliseconds, perhaps even a few seconds. It can be triggered typically at flow creation time or at times when considerable changes in the context of a flow happen (e.g., user movement).
- NetInf mostly acts on short timescales, on the order of data requests (few milliseconds, "line speed" in the sense of objects not packets flowing through the NetInf system). Opportunistic actions (e.g., deciding to add an object to a cache through which it traverses anyway) are taken at the same timescales. Management and organization actions (e.g., adding nodes to a

DHT (dht) system, starting a new cache process in an in-network data center) can well profit from CloNe and are hence linked to the timescales of CloNe.

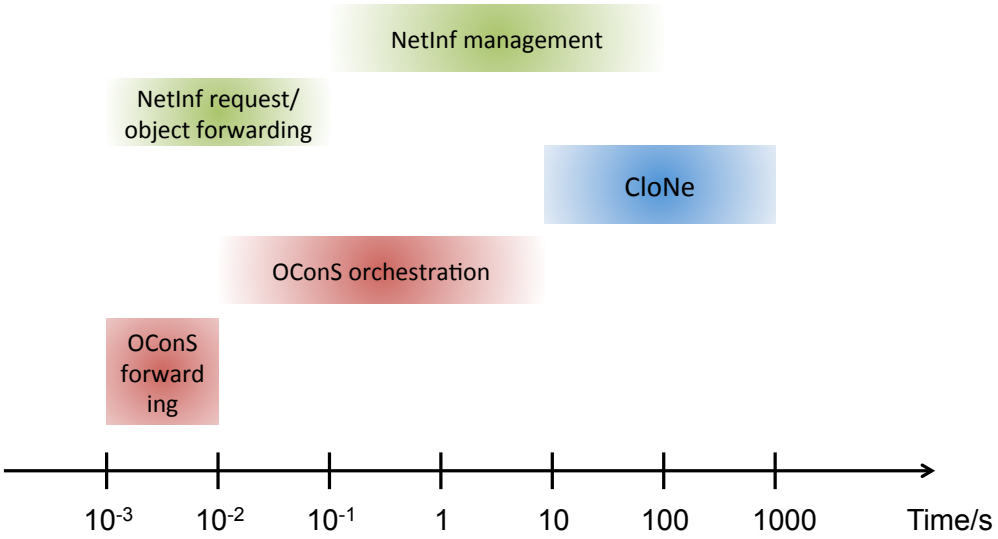


Figure 2.2: Timescales of the SAIL components

3 Interaction between NetInf and CloNe

When looking at the interactions between NetInf and CloNe, we observe the following:

- From the perspective of NetInf, CloNe provides an execution environment for an "intermediate" deployment. While ideally NetInf would be natively deployed on each and every router in the Internet, this goal seems non-viable in the near future. A deployment as pure overlay or overlay plus some NetInf-enabled router hardware is much more realistic. Here relying on data centers is a good idea as those are typically located close to the core of a network which suits well for an initial deployment. Moreover, the current trends on virtualising the network functions (see NFV [9]), smaller data centers, or virtualised resources will be available closer to the edge of the networks allowing a more suitable distributed deployment of NetInf nodes.

  Using CloNe for managing such a data center overlay deployment brings an additional advantage: the NetInf system can dynamically grow and shrink. While this is also possible in bare metal environments, it is much more time-consuming and costly – e.g., installing a NetInf module on a router, or installing new hardware.

  In such an environment NetInf needs to be able to auto-configure as it dynamically scales along with the CloNe-provided resources. While this is a requirement for "native" NetInf operation anyway, some NetInf prototypes need to be adapted to this requirement. What is new for a NetInf on top of CloNe deployment is the need for flexibility and the ensuing capability to react to changing load patterns by growing or shrinking the number of running NetInf components in a given NetInf system. This can be achieved by monitoring Online Performance Indicators (OPIs) in the running system. Moreover, using elasticity of CloNe allows to scale the NetInf system to use and pay for resources in relation with the actual demand.

- From the perspective of CloNe, NetInf is an application with some specific communication requirements (see D.D.3 [2] for a detailed account on how CloNe deals with applications in general). That means a NetInf node (say, a router or in particular a cache) can run in CloNe. Since CloNe provides network connectivity to other components of an application both inside a slice and to the outside world, the two important cases of NetInf connectivity can be solved: A NetInf node running inside a CloNe slice can talk to other NetInf nodes running
  - **inside** the same slice, taking advantage of CloNe networking facilities.
  - **outside** this CloNe slice, for example natively deployed on router hardware somewhere in the network or even on end systems.

  Hence, CloNe supports a **mixed deployment** of NetInf: Parts of one NetInf setup can run natively on real hardware routers or end systems, parts of it can run inside CloNe. This realises the required flexibility to dynamically scale NetInf on demand.

3.1 Complementary Functions

NetInf is, in particular, a fairly interesting application for CloNe since it can considerably profit from having its nodes run at cleverly chosen locations. The challenge here is that neither CloNe alone nor NetInf alone can decide on optimal locations to start a new NetInf node.

Consider the following example (a simplified version of the "Elastic NetInf" use case): Two domains X and Y are hosting CloNe environments. The domain Y is connected to a user population of substantial size. A NetInf node (cache, name resolution) is also running in domain X (natively or inside CloNe, that does not matter) that serves the user population connected to both domains X and Y. Ideally, another NetInf node should be started at Y (in Y's CloNe environment), connect to X's NetInf node, and start serving the user population connected to domain Y.

This ideal choice cannot be made by NetInf alone, since it has no knowledge that at Y, a CloNe site exists that could host such a NetInf node. It can also not be made by CloNe alone, since CloNe has no knowledge whether it would actually be useful to start up such an additional node, since it has no understanding of the consequences for the traffic flows inside an application like NetInf.

To overcome this impasse, we have created the concept of a **steering daemon** (or simply steering): It can run concurrently with an application (like NetInf) and can exchange with CloNe information on application load, topology requirements and application preferences. Both the application and CloNe can then take useful actions in response to the received information.

In the NetInf example, the steering daemon runs inside a NetInf VM, accepts topology change information from CloNe and translates this into updated configuration information for NetInf (e.g. it writes a new version of a configuration file that triggers NetInf to reread it and change its behavior – alternative actions for this daemon could be to trigger an auto-discovery process, etc.). Such a daemon builds the bridge from CloNe to a *CloNe-agnostic* application like NetInf and can be easily custom-tailored for each new application that is to run in CloNe and is topology-sensitive. Making a distributed application like NetInf CloNe-aware requires some additional steps: NetInf itself needs to know about the possibility to request more resources (e.g., triggered by observed load); details for the call sequences in this case are more complex and will be detailed in the coming sections. Put briefly: there are calls from the application to CloNe which specify load information (level of load, source of the traffic, ...); CloNe can send events like "change of topology" or "shut down" to the application which is expected to react accordingly; applications can provide plugins as decision logic to react to load information.

While experimenting with application elasticity, we conclude that some additions were required to the CloNe architecture as described in D.D.3. Those additions as well as the steering daemon are described in Section 3.3. It is important to point out that this is a quite general concept and is not limited to NetInf as a distributed application (partially) using CloNe – in the spirit of independently deployable subsystems of SAIL.

3.2 Commonalities

One could construe a possible overlap and replication of functions between CloNe and NetInf in the following fashion: Both subsystems create communication topologies on the basis of underlying connectivity. CloNe creates the network slice, NetInf creates its name resolution topology and its caching topology.

Similarly, in both subsystems, there is a need for discovery mechanisms. In CloNe it is about discovering neighbouring domains and available resources in those domains. In NetInf, discovering nodes participating in the name resolution system or in collaborative caches are examples of requirements for a discovery mechanisms.

While that is correct, we argue that this type of duplication is both common practice and unavoidable. The NetInf virtual topology is highly geared towards the particular ICN needs, whereas the CloNe slice constructions are much more general and rather oriented to the underlying topology.

Creating such virtual topologies is also a basic action of almost any non-trivial communication system and the recursive approach is common practice in essentially all relevant networking

architectures.

3.3 Application Elasticity in CloNe

In the *CloNe Architecture Description* (D.D.3 [2]), we present the main interface allowing the application provider to deploy elastic applications. The Infrastructure Service Interface (ISI) provides the means for the application provider to specify the virtual infrastructure sustaining the application as well as goals and constraints that must be fulfilled during its lifetime.

The description with the goals and constraints are analysed by the CloNe service layer (see [2, Chapter 2]) (using the goal translation and the decomposer functions) and the appropriate resources are allocated in the different domains. The allocated resources are then monitored by the infrastructure provider to ensure the goals are achieved and the constraints are respected. In case of divergence between the goals and the measured performance, the service provider can modify the resource allocations over time, adding or removing resources to the virtual infrastructure, migrating VMs, and so on. A reporting mechanism in the ISI allow to provide feedback to the application provider. However, very few details were given on how the elasticity could be realised.

In the implemented CloNe prototype described in D.D.2 [10], the elastic feature was left for further study. Since this application elasticity is required to realise the full potential between NetInf and CloNe, we have thus designed and experimented via prototyping the additional components and interfaces to achieve our elasticity objectives.

In this section, we examine how the CloNe architecture can be adapted to automate elasticity of deployed applications. Since we are working in the context of a distributed cloud, our objective is to improve the application's quality of service, usability, and interactivity by selecting not only the right amount of resources but also their location. By deploying application resources in the cloud closer to customers, we will minimise the latency between customers and the application resources and potentially reduce the network resources consumed.

We assume the Application Provider (AP) has a good algorithm to select appropriate data centers (DCs). For best possible closeness, the AP could deploy Virtual Machines (VMs) of each application at each DC. However, the overall VM expenses will dramatically increase. Intuitively, only VMs at DCs close to "big, important or large sets of customers" need to be allocated and paid. Important customers are for example intensive, frequently, or prioritised/premium customers. Knowing customer location and importance is key for achieving the optimisation goal, e.g., minimising the average request-weighted response time of a request. The optimisation algorithm needs the data for computing the optimal set of DCs, where application resources have to be allocated. This data may, however, be application-specific and not known to the cloud provider.

Special focus lies on the interfaces and information sharing/hiding of such an architecture that spreads across different administrative domains. Different kinds of partners (ISP, Cloud Provider, Application Provider) have to exchange information in order to optimise the scaling of the application. Decision strategies to optimise virtualised infrastructure and application deployment need to be customisable, so that individual needs of application developer/provider, which are as different as applications can be, are covered.

This document doesn't provide algorithmic details on how to scale optimally as this is in many case application specific, but rather specifies the interfaces for such algorithms. The algorithms developed during this work will be documented in a separate report.

The presented framework improves today's applications' quality of service by bringing application level information/operations and infrastructure allocation optimisations closer together.

3.3.1 Requirements for Application Elasticity

This section lists the additional requirements to be supported by the architecture. These requirements have been derived from analysis of the Elastic NetInf Deployment scenario presented in D.A.9 [7]. Some of those requirements are already fulfilled by the CloNe architecture as defined in D.D.3 [2].

R1: Single point Despite the geographical distribution of the application, the AP has one point of contact with the infrastructure provider.

R2: Application architecture Supported application architectures are single- and multi-tier architectures. A description of the application components' relationships is specified as an application template. This template description is required as the optimisation decides the application layout and VI.

R3: Decision generalisation The application layout is computed by a placement decision algorithm. The decision can depend on infrastructure monitoring data or application-specific data – the OPI. The variety of decision algorithms, optimisations, or rule-based systems is potentially as large as the variety of applications. The architecture should support a large number of possible decision algorithms and serve as a frame for future extensions. Both the decision algorithms and their input data need to be specified while requesting an application deployment.

R4: Information handling and hiding The architecture should enable collecting data from multiple layers and components, from infrastructure layer, from within data centers, e.g., resource utilisation, price, from network operators, e.g., connectivity between data centers, or from application layer, e.g., kind and number of processed requests. This wide range of data as input for the decision algorithm enables a new potential in distributed cloud resource utilisation. Section 3.3.3 provides not only a closer look at the different sources of information but also discuss approaches to cope with confidential information.

R5: Elasticity support The application is faced with an ever changing application layout and VI. New application instances join/leave the application layout. New application instances and VMs are started or stopped at different locations. The architecture needs interfaces to exchange information to enable applications to dynamically (elastically) connect together the different application instances.

R6: Multi-tenancy The architecture should mediate the available resources between different application providers. Individual application layout optimisation might be improved if competing decision algorithms (implicitly) exchange information, e.g., costs of resources.

3.3.2 Architecture for Elasticity

This section presents a refined high-level architecture of CloNe, introducing the Application Deployment Toolkit (ADT) as the key component in the CloNe architecture for supporting dynamically adapted applications. ADT serves as the intermediate between the infrastructure provider and the application provider, translating the application requirements into allocated virtual infrastructure. This will be described in more details in the coming sections by first revisiting some of the concepts introduced in the CloNe architecture.

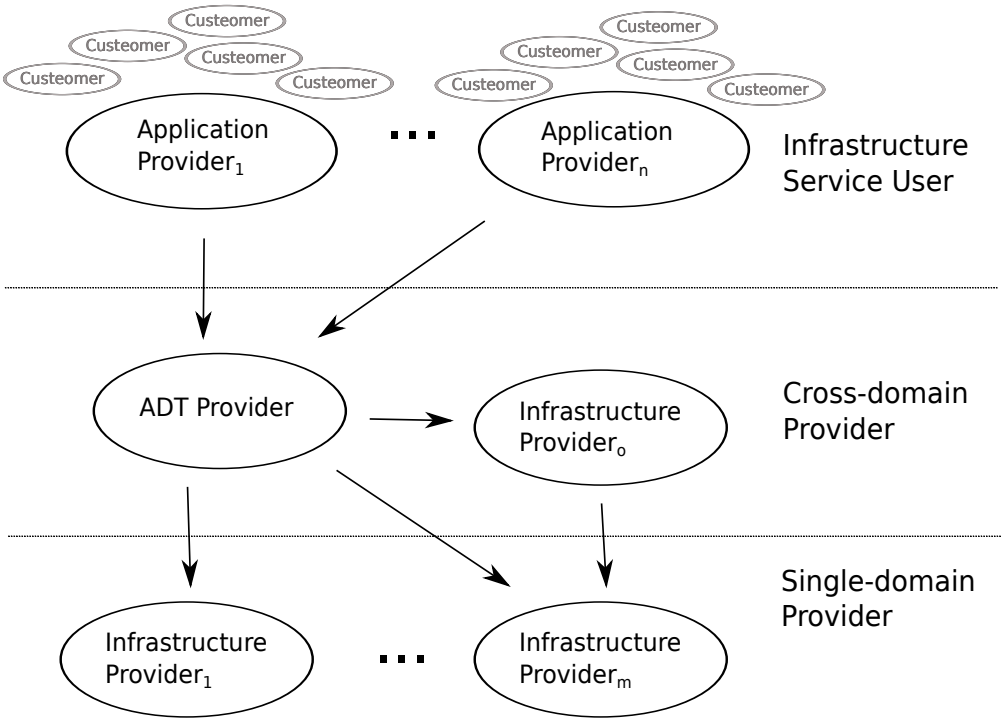


Figure 3.1: Different actors with ADT.

3.3.2.1 CloNe Roles and Layers

Roles Our CloNe architecture describes three roles (D.D.3 [2]): the user, the single-domain infrastructure provider, and the cross-domain infrastructure provider. The difference between the latter two providers is that a cross-domain provider is able to delegate requests to other (single- or cross-domain) providers. As a concrete goal translator and decomposer ADT's provider is a cross-domain provider utilizing resources from different CloNe providers. To simplify the following text, we use ADT provider as an synonym for a cross-domain CloNe provider operating ADT. Such a provider is also a distributed infrastructure provider as it requests resources from different sites and does not necessarily own resources.

For clarity, we use application provider as a synonym for user of CloNe. The application deployed by the application provider on CloNe has customers or users of the application.

Figure 3.1 shows the roles and which actor took the active role: APs are in general infrastructure service user (D.D.3 [2]). Multiple APs request adapted application deployments from an ADT provider. The interface between an AP and ADT provider is a detailed interface instance of a infrastructure service interface and is described in Section 3.3.2.5. The ADT provider decomposes AP's requirements into low level resource requests for single- or cross-domain infrastructure provider.

Layers Figure 3.2 show the four layers of CloNe (D.D.3 [2]) extended by an application-related layer. An elastic application is a distributed system which is composed out of components (shown as red squares) at the application layer. They not only communicate with each other, but also provide application-level monitoring data, which are used by ADT. ADT itself is able to reconfigure the application and supports application elasticity (steering concept). This bidirectional information exchange is one of the major contributions of this architecture and is described in Sections 3.3.2.2 and 3.3.2.4. It allows a tighter connection between infrastructure and application.

ADT is an extension of the goal translator and decomposer functions. Unlike the previously described goal translator, ADT not only utilises infrastructure but also application-layer informa-

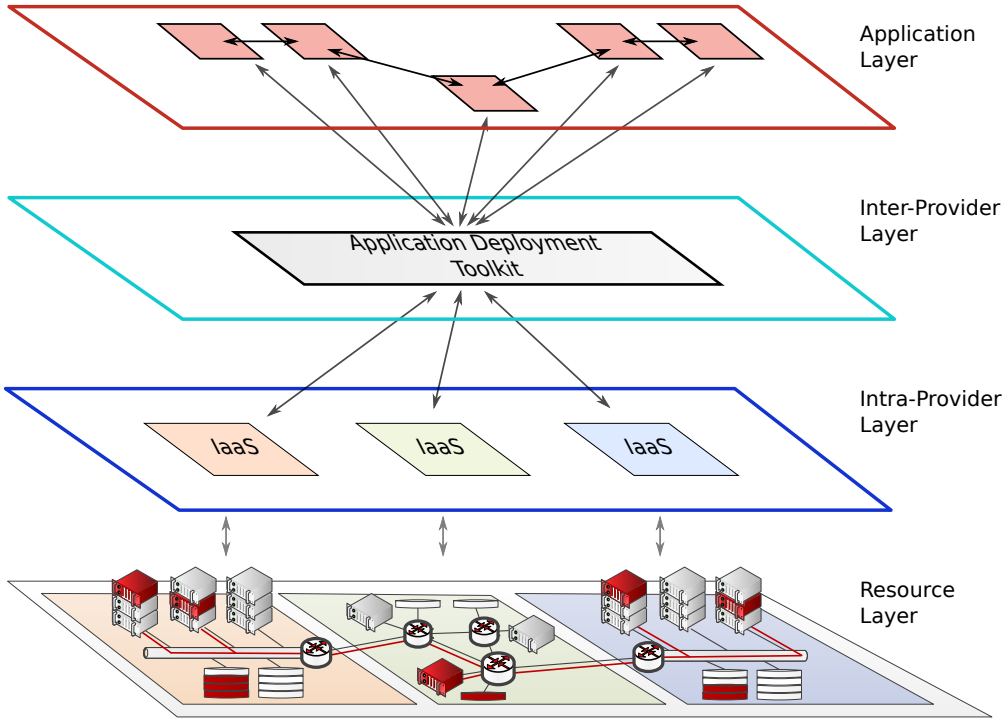


Figure 3.2: CloNe Layer and information sharing across layers

tion to optimise placement. ADT also encompasses a concrete approach towards a geographically distributed application management (steering) and a concrete concept of how to compute and implement an optimised placement of VMs. ADT utilises topological information for such an optimisation.

The ADT provider interacts with other CloNe Infrastructure Services as a Distributed Infrastructure Service (D.D.3 [2]).

3.3.2.2 Additional Functions and Interfaces

To realise the elasticity in coordination with the application, CloNe requires the following new functions and interfaces.

Interaction with the application (Steering) The ADT approach is to share information between infrastructure management (ADT) and application. This is applicable in three situations:

(a) to inform the application before or after an infrastructure operation takes place.

(b) to establish a communication system between AP and application's instances.

(c) to report application level or in VM monitoring data to improve decision implementations.

Both application and infrastructure provider might want to hide confidential information. Section 3.3.3 describes applicable approaches to such situations.

Information retrieval Information is essential to make good decisions. There are three levels of details of information that can be collected and provided to make allocation decisions.

1. Information about available cloud resources from the infrastructure providers. This includes topology between the domains, class of resources (compute, storage, network), data center geographic or topologic location.

2. Online monitoring information of a deployed application's virtual infrastructure.

3. Application-specific performance indicators and metrics. This value can only be collected by the application. This data is denoted by the OPI and can vary depending on the application or optimisation goal.

Generalised Decision Strategy To provide a flexible and generic framework for resource placement, the architecture provides a placeholder for the decision strategy.

This introduces an orthogonal layer to support different kind of configurations of decision modules (DM). The interfaces will have placeholders to support a wide range of different decision modules. The two placeholders are: The decision module configuration `custom-dm` and the input data - the OPI - `custom-opi`. We can envision three fundamentally different kinds of decision making strategies:

specialised The ADT Provider creates on behalf of the AP a narrowly tailored decision algorithm optimising the application's VI for a very special or single application.

configurable In this option, the ADT Provider offers optimisation goals for known application deployment schemes, e.g., multi tier application to be deployed and optimised for low latency. Each of these decision algorithms can support a wide range of application, application architecture, optimisation goals, multiple goals, or parameters (e.g. front-end to back-end server ratio). Also they can have their specific configurable optimisation parameters (latency, costs, ...).

custom The AP might have its own decision logic for a special application. Such a logic can be provided as a whole customised DM. Such a DM can be plugged into ADT and is invoked by ADT to manage the application's VI.

3.3.2.3 ADT Architecture

This section discusses the trade-offs between a centralised and distributed technical architecture of the ADT.

Centralised The simplest solution is a centralised ADT component, e.g., a set of machines located at one site. Beside the downside of being a potential bottleneck, the collected data provides a global view and a much simpler non-distributed decision algorithm can compute placement optimisations. In addition to this benefit the architecture design is also simpler.

Distributed The major downside of a central architecture - the bottleneck - is relaxed if a set of geographically distributed ADT components work together as a distributed system. Therefore, the overall ADT covered topology is partitioned so that one ADT component hands information exchange, decision logic, and VI changes at its partition. This approach has two major downsides: First, the decision algorithms have to be designed as distributed algorithms and have to handle only a subset of the overall application layout. Second, the ADT architecture itself becomes much more complex.

However, for a first proposal of an architecture encompassing all requirements, the centralised approach promises to obtain first prototyping and algorithm results faster. The concepts should also be applicable and extendable to a distributed ADT architecture.

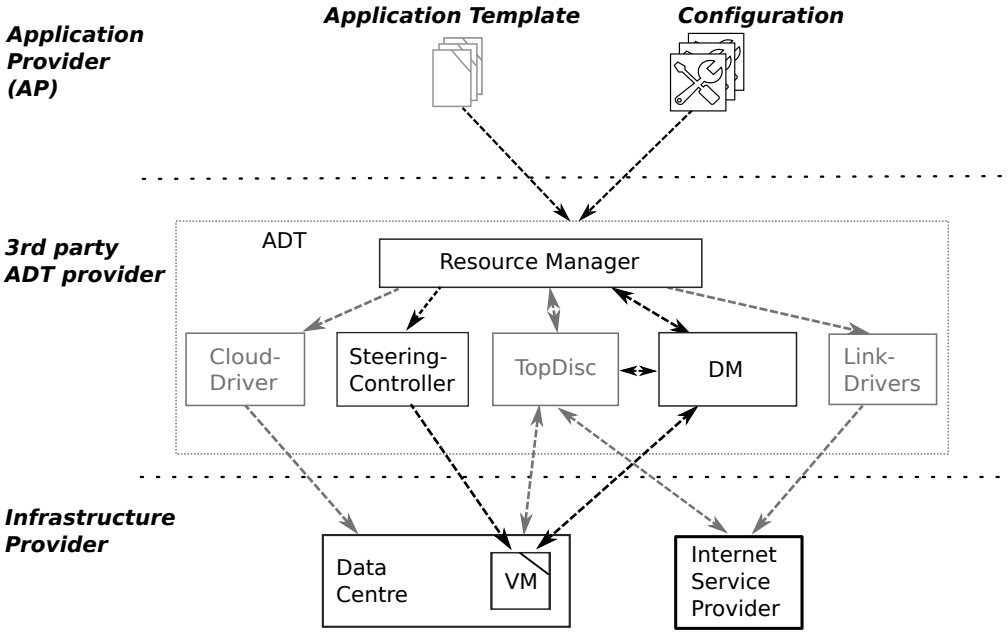


Figure 3.3: The detailed ADT architecture

3.3.2.4 ADT Detailed Architecture

The following ADT modules illustrated in Figure 3.3 are involved in the application requests and elasticity processing:

Resource Manager decomposes the abstract control operations into low level operations to individual IP . Therefore it maintains the current application layout and VI and the changes needed for the abstract operations. It also coordinates the application support for elasticity by notifying the application instances before or after infrastructure operations take place. Therefore, it utilises the tool modules Steering Controller, Cloud Driver, and Link Driver.

Steering Controller takes care about sending data or events to the application instances.

Cloud Driver handles the different cloud computing interface: OCCI (and flavors), EC2, libCloud, etc.

Link Driver handles network related actions:

- datarate-reservation between individual application instances or between sets of application instances at different data centres
- DHCP configurations
- directly accessing SDN.

Decision Module takes the decision on resource placement and changes to the virtual infrastructure. The DM has three input sources:

1. Topology Information can be queried from Topology Discovery Module (TopDisc);
2. DM can query VI information using the *list* and *status* commands.
3. The application and ADT can provide update OPI values using a *report* message.

The decision algorithms' result could induce three potential actions:

1. The DM changes the virtual infrastructure: network, storage, or compute resources have to be added or removed; this is done through *change-elastic-vi*.
2. The application layout is changed even if the VI remains the same; this is applied through an *update* command.
3. Application level configuration needs to be changes. Application's instances are notified by an *update* command.

The DM can provided by the application provider or by ADT, in which case the AP will select the DM from the set available at the ADT and will configure it to its specific needs.

VM The virtual machines have a steering daemon running, which is responsible of interaction with ADT, receiving notifications on the life cycle of the VM through the Steering interface, and reporting events and measurements to ADT through report interface.

3.3.2.5 ADT Interfaces

This section describes a detailed description of an Infrastructure Service Interface described in D.D.3 [2]. As an Infrastructure Service User the AP can express changes of the VI, e.g., more/less components at a data center, more condensed. The deployment, configuration, and connection setup of the different components are done automatically in the background. Upon request the application is specified as a whole entity in an **application template** containing the following information:

Application Architecture : A graph describes the application components and their relationships. This is used to properly connect the different component.

Deployment Requirement : Nodes are annotated with deployment information: VM image (id, url), resource requirements (cpu, mem, additional storage, etc.)

Communication : Nodes are annotated with custom data used during initialisation. This enables the AP to customise a VM and containing application's instance at request time, e.g., with fresh credentials.

This part of the application interface is inline with the interface defined in D.D.3. Without going in details in describing all interfaces involved, we give here an overview of two of the interfaces implemented in the experimental system: ADT and Steering.

ADT commands The ADT interface contains the following commands:

smart-deploy Request/create a new application deployment, not interaction using steering and dm are expected. This is the interface that

smart-elastic-deploy Similar to the *smart-deploy* request but extended with parameters for the elasticity decision process. Deployed application components are expected to interact with the steering daemon.

list Query an overview of all application instances.

status Query status of application instances.

change-vi New components can be deployed or removed.

change-elastic-vi Similar to the *change-vi* request but provides additional data for the elasticity decision process.

destroy The whole deployment can be stopped.

checkpoint Checkpoint the state of all or only a subset of the application instances.

suspend Suspend a set of application instances.

resume Resume suspended application instances.

resize Change resource allocation of application instances.

broadcast Broadcast information to application instances.

evoke Evocation of custom function at application instances.

config-dm Configuration of the decision module.

In the automated elastic deployment, the Resource Manager interacts with the Decision Module. When a VI is instantiated the DM needs to be allocated and configured using the information received in the *smart-elastic-deploy*. This is done using textitinit-dm function in the DM interface.

Steering The steering interface encompasses the communication between infrastructure and the application level to indicate events in the life cycle of the application instances. These event notifications allow the application to set or update configuration, react properly to preserve data integrity or adapt behaviour to new network or resource situation.

The interface includes acknowledge messages that allows the application instance to indicate the ADT that it has completed to process the event.

The events considered in the interface are:

on-boot This message when the application instance both on initial boot as well as on reboot.

on-resume This message is sent after a VM has been resumed. It allows the application instance to synchronise state with other application components.

on-suspend The message is sent before application instance is suspended. It allows the application instance to save its state or delegate its role to some other instances.

on-conn-update This event is sent when the connection situation of the application instance has changed.

on-destroy This message is sent before the application instance is destroyed.

on-resize ADT sends this message when the resources allocated (cpu, mem, ...) to the application instance are updated (increased or reduced).

broadcast This message is sent to all application instances by the ADT upon request from the application.

3.3.2.6 Control Flow

This section describes the sequence of actions with the extended architecture including ADT. The sequence diagrams in Figures 3.4 and 3.5 show the initial deployment and the decision module reaction of changed OPI. These diagrams are simplified by two aspects: The Infrastructure Provider requests are not shown and Tool modules, the steering controller, the cloud and network driver are not shown.

We use a very simple application that tracks the source of its users. The application instances trace their customers location and report them to the DM. The placeholder `custom-opi` sent from the application instances to the DM contains a list of source locations of requesting customers. The DM selects based on the locality and intensity of all customers, where to place VMs to optimally serve the customers. The closeness of locations are queried from the topology discovery module (TD). The placeholder `dm-data` is a mapping for each customer to VMs and is computed as part of the result of DM's optimisation. This data is sent to the VMs/instances/peers to redirect wrong accessing customers.

Initial deployment

1. AP deploys the application with a decision module configuration included:
   `smartly-elastic-deploy` (a).
   `dm-id` selects a DM and `dm-config` contains its configuration.
   `dm-data` (in `alloc` for the initial allocation) is empty, thus customers are not redirected.
2. The RM instantiates a new DM and initialises it, `init-dm` (b).
   It additionally acknowledges request (a) and continues with the initial deployment.
3. The initial deployment is allocated and the VMs are configured: `on-boot` (c, c').

Monitoring The monitoring loop describes a reoccurring process of tracking reported OPI and starting a processing of the updated OPIs. The sequence is described in 3.5.

1. Active VMs continuously report their OPI, `report` (d). These reports are received at any time.
2. Before processing the updated OPI, the topology model is updated, `get-top` (e).
3. The processing (f) of updated OPI.

Apply Results This paragraph shows the realisation of a changed decision-placement as example:
1. The old deployment (VM_1, VM_2) is changed into (VM_1, VM_3) thus a VM has to be shut down and another at location 3 have to be started. Of course another DM could have decided to simply migrate the VM.
2. Decision module data `custom-dm` – the mapping to redirect customers – are send to the VMs.

1. DM recognise a new result and informs RM to change the VI `change-vi` (h).
   Additionally, DM let the DM take care of sending `dm-data` to VMs.
2. After requesting a new VM, the VM is initialised with `on-boot` (i).
   `dm-data` is set for this new VM.
3. After receiving the response, the other dependent VMs are notified about the new VM `update` (j,j').
   `dm-data` is set only for VM_1 as VM_2 has to be removed.
4. Immediately after all new VMs are running, the VM_2 is notify about it's future shut down: `on-destroy` (k).
   This is done after integrating the new VM, so data have more location to be backed up, eventually directly to the new location.
5. After the `gracetime` is passed, RM first notifies the VM about the phase out of VM_2 (`update`) and then destroys the VM.

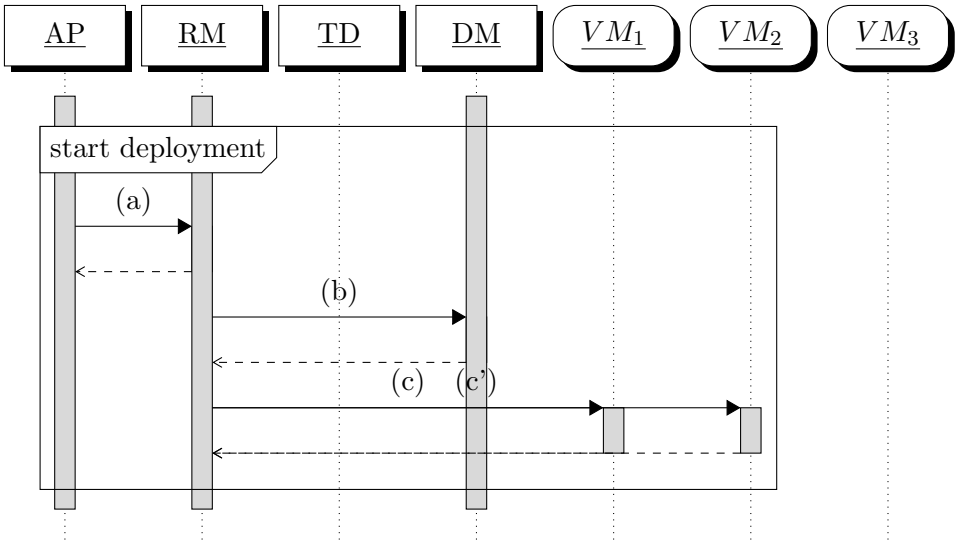


Figure 3.4: Initial deployment of adapted deployment.

Role	Layer	Deployment	Monitoring
IP	single-domain	available resources resource granularity	host utilisation energy consumption network utilisation network topology
DIP	cross-domain	connection graph (IPs)	
AP	application-level	software architecture application scalability	(weighted) requesting rates internal data: queue lengths VM data: storage, memory, cpu utilisation network device's data rates

Table 3.1: Interesting data affecting application's VI layout.

3.3.3 Information Handling

The core functionality of adaptive deployment is implemented in the decision module. This module computes a new VI layout. Such a computation is based upon information, monitoring data, or input data. Usually, such a computation solves a mathematically problem, e.g., using the input data to minimise expenses, improve quality of service or stability, ... A complex problem expresses complex relationships by integrating more input parameter, e.g., cpu utilisation, resource cost. Optimisations of such complex problems have a greater potential of quality improvement or cost/expense reduction as simple ones. In summary, the more information DM can process, the greater is the module's potential.

Such data ranges from single-domain over cross-domain to application-level information. Data can be dynamic and can be retrieved or monitored during runtime of the system. Other more static data relates to deployments (and VI changes). Table 3.1 lists some examples sorted by type of data and layer affiliation, e.g., the IP provides single-domain data or the AP provides application-level data.

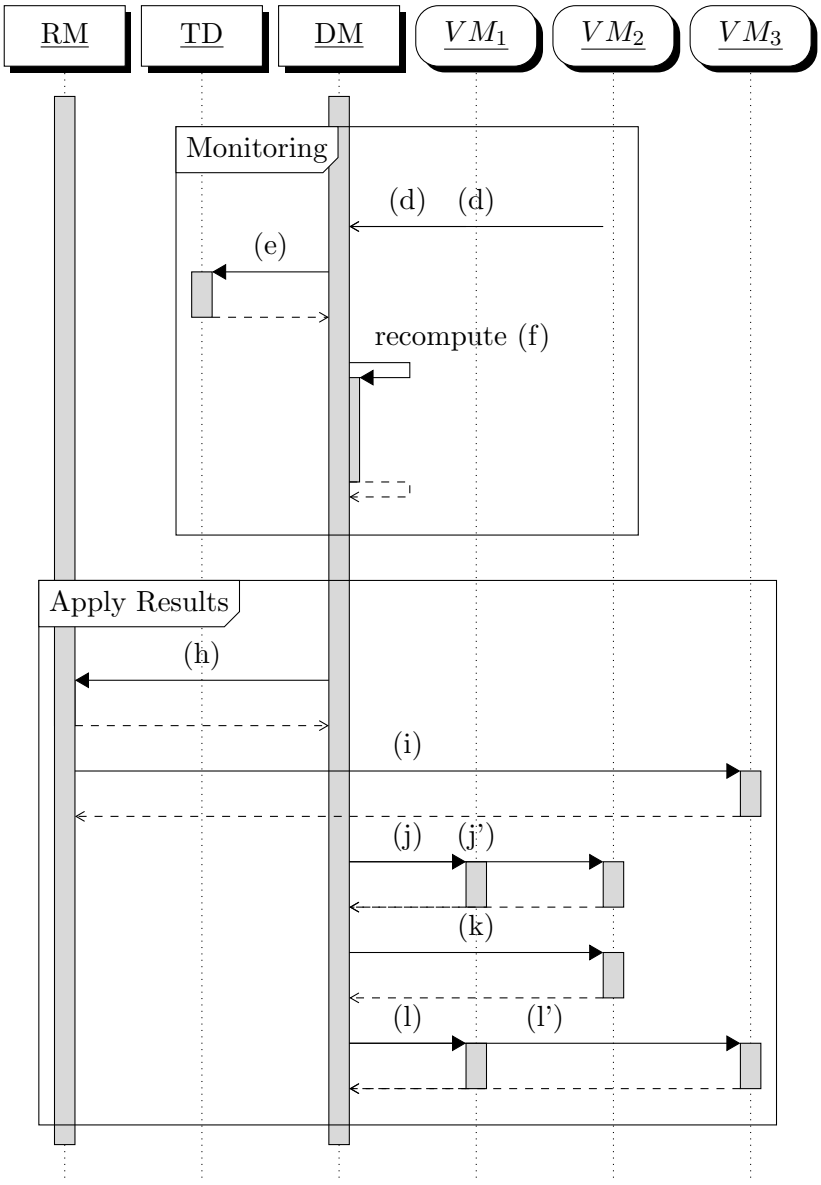


Figure 3.5: Adaptation of decision module

Issue The presented architecture allows to accumulate all these data to a single spot – the decision module – and thus enables maximising the potential improvement for complex optimisations. But this also transfers data sets across layers and domains and internal data might be unintentionally exposed. Two adversaries to cope with are identified: (a) the ADT provider exploits the data accumulation and (b) an AP miss-use the system to get information. Finally, the implication of not sharing internal data at all is discussed.

ADT provider Hosting ADT enables to spy out ADT's data and therewith all accumulated data, for instance from competing IPs. This can be done by manipulating ADT's functionality, accessing ADT's memory, or tracking ADT's communication. In short the ADT and its operating system is compromised by an adversary ADT provider. A way out is offered by Trusted Computing (TC)[1], which guarantees ADT's peers – the customers, APs, or IPs – that the ADT system is not compromised and works without manipulation. Thus an ADT provider is not able to access or manipulate ADT without being noticed by the other peers. Secure communication prevent tracking of send data. However, the root of trust – that this security system works – is a tamper resistant Trusted Computing Module on the motherboard and a key-pair stored within which is issued by a trusted 3rd party.

Application Provider The ADT provider is trusted to keep data confidential, e.g., by TC or contract. This reduces the requirement to assuring that confidential information are not spied out by an adversary peer: AP $\leftrightarrow$ AP $\rightarrow$ IP $\leftrightarrow$ IP. Each of the APs have their own DM, are treated independently, and the access can be secured normally (multi-tenancy security). AP can send application level data similar to their requests encrypted and their VMs are isolated. As a result, AP can't hijack other AP's data (AP $\leftrightarrow$ AP). The ADT interacts with IPs in two ways: (a) The resource manager deploys and allocates resources and (b) the topology discovery module requests topology and monitoring data from IPs. In both ways ADT interact with each IP individual. As a result, IP can't hijack other IP's data (IP $\leftrightarrow$ IP). Consequently, information are only merged inside ADT. For example, the TD module provides an unified view to resource properties including for example price, capacity, and topological location. The DM processes this data. Its configuration can either be (a) programmed or (b) configured. (AP $\rightarrow$ IP)

The first most flexible way allows the AP to upload code for his DM. The code retrieves input data from internal ADT interfaces to process it and compute a new placement. But however, an adversary AP can program its code to additionally send this internal input data around. To fix this leak for programmed DM either (a1) the ADT provider has to verify adversary AP's code to not send infrastructure data around or (a2) the option for a programmed DM has to be disabled.

In the second way, an AP configures DMs offered by an ADT provider. The ADT provider has to verify that all possible configurations do not leak information. An adversary AP can't configure DM to leak information (AP $\rightarrow$ IP).

Degree of sharing The IP can keep any confidential data. The leading ideas is that the securest way to keep information confidential is: do not share it. However, as a result of missing information ADT, the DM, and their optimisations are hampered and this result in a worse placement and more costs for the ADT provider or AP. In general, it is beneficial for IPs to share data, because among equal IPs the one who cooperates more might have a competitive advantage from the AP point of view. Eventually, some services like in-Network cloud resources can only be efficiently, dynamically utilised by ADT with more information about their topological location – this information is a typical example of confidential information.

[1]Trusted Computing Group Website: `http://www.trustedcomputinggroup.org/`

The alternative for a cooperative IP to not share all details is to provide them as an aggregated or abstract value: The cloud resources have an approximate topological location or real values can be discretised or aggregated. This hides the real topology or infrastructure data while providing the decision logic an increased level of accuracy compared to no topological data.

The degree of level of detail provided by an IP controls the trade off between (a) how many information kept confidential and (b) how good the results of optimisations are.

Summary ADT can cope with an potential untrusted adversary AP either by restricting the DM configuration if the AP provider is trusted or the IP provides abstract information for an untrusted AP provider. In general, information can't leak between APs or AP's DMs or between IPs.

3.4 Elastic NetInf

Based on the discussion of CloNe elasticity in general, let us see how this applies to NetInf on top of CloNe in particular.

NetInf's VI is adapted to meet changing demand while the quality of service remains similar. A typical use case is a content with rising popularity. This results in (a) higher usage intensity at (b) different requester's locations. Today's NetInf replaces less popular content in caches by more popular content. This reduces the quality of access for such dropped content. In such a situation, increasing the cache capacity is desirable. We leverage the potential of cloud infrastructure's on-demand provisioning feature to swiftly provide NetInf with new resources at appropriate locations. This enables NetInf to handle content with rising popularity without reducing the access quality of the other content.

In addition, we utilise on-demand infrastructure resources to improve NetInf coverage. Resources at new locations can handle requests more locally and more caches will increase overall system performance.

Summarising we have two use cases for an elastic NetInf deployment:

Adapt at fix locations: NetInf's locations are fix or manually maintained. ADT increases resources of NetInf nodes or deploys new nodes in VMs at the same location/data centre. Therefore, the ADT's monitoring-reconfiguration-loop exchanges Online Performance Indicator (OPI) about the popularity of requested content. Based on this information ADT's NetInf Decision Module reconfigures the resources utilised by NetInf.

Expand to improve coverage: Extending the previous use case, NetInf nodes are deployed to new data centre locations. This introduces a new problem to be solved by the Decision Module: How to select the best location among available data centres? Which information is necessary to answer this question and which is available?

This section covers the first use case by describing the OPI and possible reconfigurations. Afterwards, the challenges of the second use case are discussed in Section 3.4.2. Finally, an approach to integrate Netinf's performance indicators is presented in Section 3.4.3

3.4.1 Adapt at Fixed Locations

This section describes two parts to enable the adaption: (a) the monitored data and (b) decision logic. From a top level, resource allocation is decided based upon requesting rate of content and request location.

Let's assume for now (to simplify the discussion) that NetInf is deployed using the Internet Protocol.

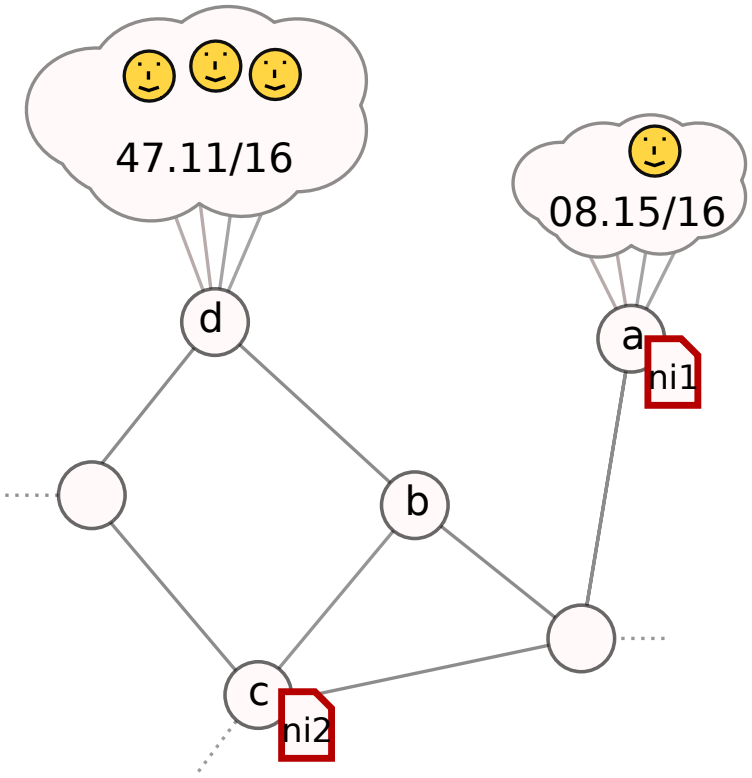


Figure 3.6: NetInf deployment blend over topology.

Monitoring Every Netinf node sends its monitoring information – NetInf's OPI – to ADT's Decision Module (DM). This DM implements the decision logic and reconfigures the NetInf system as described in Sections 3.3.2.2 and 3.3.2.6.

Every Netinf node keeps track of incoming requests for Named Data Objects (NDOs) and send this **Usage Report** to the DM. We present three versions of this **Usage Report**:

Basic The basic report is a list: [`(region, NDO, count)`,...]. `region` specifies the source of the requests, e.g., the Internet Protocol address. `NDO` is the name of the requested NDO. `count` is the amount of incoming requests of this `NDO` from `region` since the last report.

Such a list contains the most detailed data, but every report will be large.

Aggregated To shrink the report, some entries can be aggregated: `region` represents a super-set of tracked sources. This aggregation has to preserve closeness to some degree. In case of Internet Protocol addresses, subnets can be used, e.g., `12.34/16`. `NDO`s can be grouped, e.g., hierarchical names allow a natural aggregation. `count` is the sum of aggregated requests.

The degree of aggregation controls the trade-off between (a) detail information in a long report and (b) aggregated data in a short report.

Filtered The Usage Report can be further reduced by filtering it before sending. As a simple example, all entries below a threshold could be dropped.

From a conceptual point of view, a part of the decision logic is migrated to the NetInf nodes. Because of this, the filter has to be carefully designed to not accidentally hide information which would otherwise result a different deployment decision.

netinf	*region*	*NDO*	*rate*
ni1	08.15/16	xyz	1
ni2	47.11/16	xyz	3

Table 3.2: Usage Report of NetInf nodes.

top.-node	*region*	*distance*
a	08.15/16	0
..	..	..
b	08.15/16	1
b	47.11/16	1
..	..	..

Table 3.3: Region Table of topology nodes.

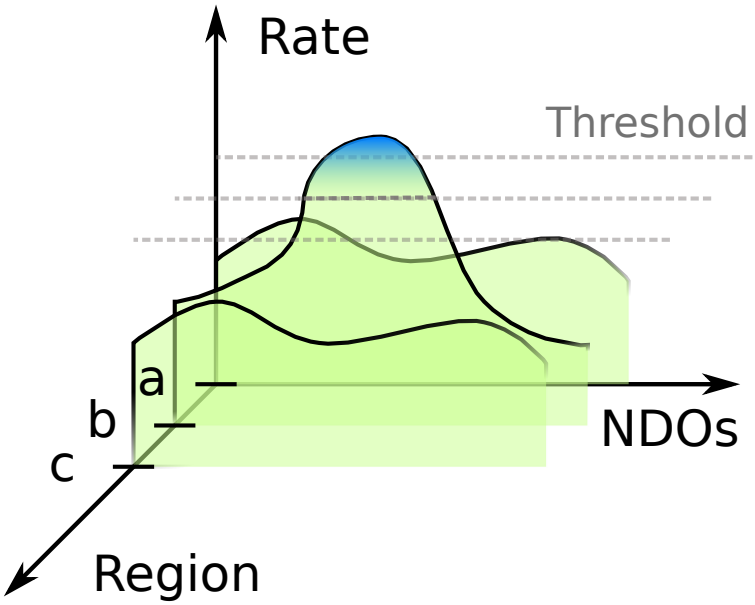


Figure 3.7: A segment of NetInf requesting rate statistics.

Figure 3.6 shows an example topology provided by ADT's topology discovery module (TD). NetInf is deployed in two VMs running at data centre nodes a and c. Two user populations are shown in the subnetwork clouds. Table 3.2 shows the corresponding reports.

The notion of locality is expressed by "distance to region" and is stored in a **Region Table** at each node (Table 3.3). Nodes can be data centres or even routers, depending on the information shared by the IPs. The subnetwork with subnet `08.15/16` is connected to node a. Node b is distance 1 away from node a and region `08.15/16`. Distance can be, e.g., hop count or latency.

Reconfiguration To simplify description, we discuss a centralised approach. Nevertheless, it has scaling limits and concepts can be applied to other architectures discussed in Section 3.3.2.3.

All NetInf nodes are sending periodically their Usage Reports to the DM. Therewith it maintains a global view of incoming requests of the whole NetInf system. Figure 3.7 shows a small segment of such a view. For each region, here a subnet, the requesting rates of each NDO are shown.

In general, a cache with a high hit rate is most useful. The hit rate depends (a) on the size of the cache and (b) on a small subset of NDOs with high requesting rates. These NDOs are likely to remain in the cache and a high hit rate is the result. In a contrast situation all NDOs have similar requesting rates. No NDO is preferred over another to stay in the cache to increase the hit rate. Figure 3.7 shows two regions a and c with similar requesting rates for NDOs. In contrast at region b, a few NDOs are significantly more popular than the other NDOs.

This relationship can be exploited by increasing resources at caches near regions with significant popular NDOs. This way, resources are allocated only at beneficial locations: A locally deployed

cache with a high hit rate serves most requests locally. To identify such situations from DM's global view (Figure 3.7) a crossing threshold could trigger VI adjustments. Fixing such a cornerstone for a DM enables the design of suitable Report Table filters.

The Non-Internet-Protocol case To extend the system to Non-Internet Protocol locations, the following properties have to be satisfied:

region: A region is a location identifier. It is reasonably fix, e.g., doesn't fluctuate in a short period of time. Otherwise it distorts the statistics of the Usage Reports or invalidates the Region Tables of the topology.

A good example is the MAC address of a physical network device while a virtual MAC address of a VM can become impractical depending on the lifetime of the VM. A similar relationship holds for Internet Protocol addresses.

closeness: Locations have to be part of a path-connected topological space with a (non-cacheable) function for distances between any pair of two locations. Only this enables to search for a nearby location. However, such a search could be very extensive. A pure distance, a metric function, simplifies and speeds up such a search.

For example hop-count is a distance but latencies between hosts are non-metric (neither symmetric nor fulfilling triangle inequality). Even so, these violations usually only occur locally. Thus, latency can be treated from an overall perspective as a distance with small inaccuracies.

Such closenesses or distances are part of the Region Table.

common understanding: NetInf and ADT need to interpret the locations in the same way. Otherwise a matching or alignment of NetInf's Usage Report and ADT's Region Tables are not possible. This implies that two independent systems need to identify the same host, router, or entity with the same location.

Obviously, as an external naming scheme, Internet Protocol or MAC addresses have this property.

aggregation: This soft property doesn't have to be satisfied mathematically, but it is necessary for practical reasons: To reduce the amount of entries to be transferred and processed, Usage Report's and Region Table's region entry also supports aggregated values representing a set of close individual locations. Similar to *closeness*, this rule can be relaxed to be more applicable while accepting slightly less accurate results: Exceptional, few locations are not part of the superset.

As an examples, Internet Protocol subnets include a set of single addresses, which are often close to each other. Exceptions (to prefix aggregation) are allowed.

A counter example are MAC addresses: Effectively, they are randomly distributed across the hosts, routers, or network devices. As a result, no aggregation rule can be derived.

In summary, Internet Protocol and MAC addresses are equally, mathematically suitable. Internet Protocol addresses are able to be aggregated, thus practical applicable and thus superior to MAC. By slightly relaxing the *closeness* property, both can be used with latency as a distance.

Alternatively, in internet coordinate systems, hosts, router, or network devices are embedded in a metric (multi-dimensional) topological space and have coordinates as their locations. Closeness is guaranteed by the embedding metric, and a n-sphere can serve as an aggregation. But a careful look at the *common understanding* is necessary: Let's assume two independent systems, like NetInf and

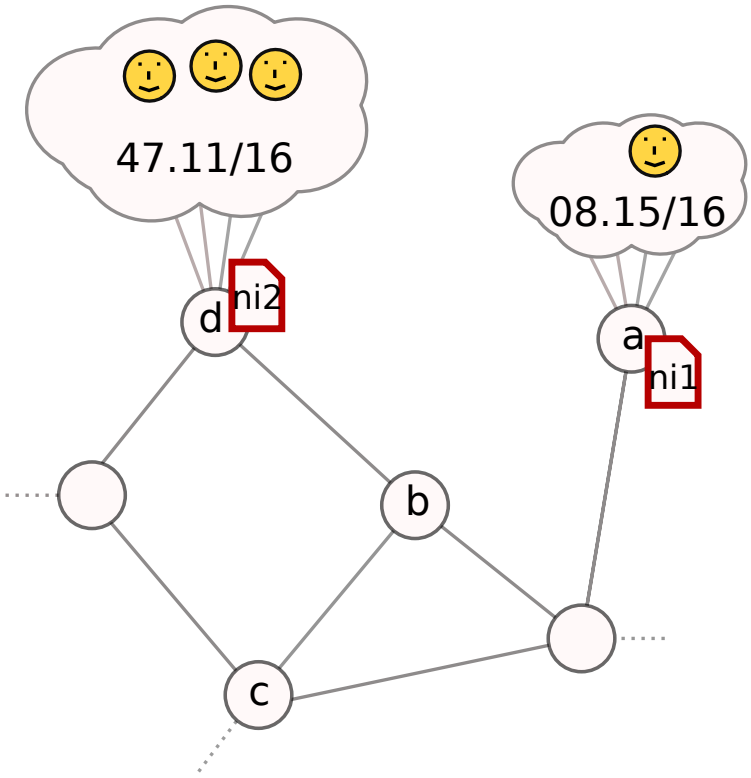


Figure 3.8: Adapted NetInf deployment.

ADT, manage their own coordinate systems. This can result in two different coordinate systems with same distances between pairs of network devices but different values (in both systems) for device's coordinate/location. Consequently, location entries in Usage Table and Region Table can't be matched correctly. For satisfying the *common understanding* property, both need to use the same coordinate system, e.g., by using the same internet coordinate service or middleware. In summary, internet coordinate systems are a promising alternative to Internet Protocol address locations.

3.4.2 Expand to Improve Coverage

Until now, the DM decides upon the resource size for each NetInf node and cache. This section extends the DM to select new locations for expanding the NetInf system.

As an example, the amount of NetInf nodes is preconfigured to 2, e.g. to limit expenses of the cloud infrastructure deployment. Looking at Figure 3.6, the global request rate view in Table 3.2 can be matched with region information from Table 3.3. Then node b is a better choice for a NetInf cache (c.f. Figure 3.8).

This example can be extended to have a trade-off between (a) cost or NetInf's service quality and (b) the amount of allocated resources. This way, not only a fix number of nodes have to be preconfigured, but the amount of nodes dynamically changes to for example optimise a quality metric: latency or data rate. The problem of selecting a good data centre to deploy a new NetInf node falls into the family of facility location problems.

3.4.3 Performance Indicators

In D.B.3 [1, Section 4.4.1] a list of Performance Indicators were drafted to give a NetInf operator indications on how well its NetInf domain is performing. Some of these indicators are suitable

to provide hints on how to adapt a virtualised NetInf system. We provide a few examples of performance indicators that were used in our NetInf over CloNe scenario to illustrate NetInf's deployment flexibility.

#NDO requests from another domain

NetInf node in a domain A provides NDOs either by agreement (publish) or because of their popularity (caching). If a large number of requests served by nodes in domain A originate from an outside domain B, that may generate a high utilisation of transport links between the domains and a sub-optimal used of resources available in both domains.

Reaction: Three possible adaptive reactions can be applied with NetInf over CloNe:

1. Requested NDOs can be made available at domain B by increasing the cache storage capacity at domain B. This may include to start a new NetInf cache if none were available previously. Therewith, NetInf can cache popular NDO's at domain B so requests are served there and not at domain A. This reduces the amount of NDO requests and the inter-domain transport link utilisation.
2. The NetInf deployment can be reorganise in domain A so that bottlenecks are mitigated. Load balancing/distribution inside a domain over (added) cache locations, Increased resources for faster computation and transport.
3. A reduction of provided or advertised NDOs will reduce the incoming #NDO requests and traffic (for example, not serving requests for NDOs not published by customers in business relation with domain A). This has the disadvantage of putting more pressure to other domains serving the NDOs.

NRS LookUp Delay

This request delay is the duration needed by the NRS system to collect a list of locators for the requested NDO and to reply to the requesting device. At least two possible issues could cause a high delay: (a) Intensive request load on a huge database of registered NDOs and (b) a distributed NRS system adds latency overhead for communication with other NRS peers.

Reaction: While issue (a) can be mitigated by adding more compute capacity to the virtualised NRS VMs, issue (b) may be harder to cope with. One approach is to change the specified characteristics (lower latency or increased bandwidth) of the networking resources requested from CloNe. This may come at a highest cost or may not be achievable at a certain point.

Storage-Capacity Cost ratio of cache location

This item is not at first view and online perfomance indicator. It illustrates the capacity of the designed system to react to changes in the environment. Operating additional NetInf caches usually comes with costs: Either plain energy and maintenance costs or contract fees for allocating resource in external clouds. While the cost of managing your own hardware are usually relatively fixed, the cost of cloud resources can fluctuate over time. For example, we have recently seen spot markets for computing resources and we could imagine that this could also happen for other type of cloud resources. The cost of cloud resources used by NetInf can fluctuate and may have an impact on the selection of cache or node location.

Reaction: A decision module can now deploy caches at beneficial cloud resources. The fundamental question asks for which cloud resource/cache location and involves cost per storage capacity, estimated neighbouring link congestion for a placed cache, estimated incoming traffic flow, etc. This can be expressed and solved as a flow problem and the solution can change as the cost of resources varies over time.

3.4.4 NetInf System Reconfiguration

CloNe's ADT architecture enables two additional ways to configure NetInf:

Connection The decision module understands NetInf's software architecture. This enables to provide connection information to new NetInf nodes. Then these nodes not only know how to connect to the running NetInf system, but also know which are their peering nodes, e.g., which are nearby nodes, their corresponding NRS, etc. In the case of a virtualised deployment of NetInf using the Internet Protocol, each new NetInf instance will obtain a dedicated Internet Protocol address that may be dependent on the network topology or the location where the instance is created. NetInf can thus benefit from being assisted by CloNe's steering events to be informed of the established connections (see Section 3.3.2.5).

Reconfiguration A decision module is able to send data to individual NetInf nodes (`custom-dm`, Section 3.3.2.6). Therewith individual node's property can be configured from an external perspective: caching strategy, reroutes, link avoidance, etc.

Native NetInf Orthogonally to these opportunities, NetInf's self-managing, self-configuration, self-optimising features (D.B.3 [1, Section 4.4 Network Management]) can co-exist. For example, NetInf can find neighboring nodes and discover topology by its own without expecting the notifications from CloNe.

In summary, CloNe's reconfiguration capabilities can be additionally used to NetInf own self-adapting features.

3.5 Summary

Virtualised NetInf on top of CloNe realises a deployment without the need of high upfront investments for exchanging hardware. Having cloud resources available today, our work enables a distributed deployment of NetInf in a very near future. This shortens the time to evaluate NetInf in a wider deployment and eases future migration steps. In addition CloNe offers network connectivity service and load-adaptive deployment. While the former enables NetInf to have guarantees for management communication and content transport, the latter allocates that appropriate amount of resources at geographically distributed sites to optimise NetInf's performance. The contribution of the CloNes Application Deployment Toolkit (ADT) is two-fold: Firstly, it refines the CloNe architecture, providing a better understanding on how customisation of a wide range of application including NetInf can be achieved. Its steering concept introduces a new exchange of communication between infrastructure and application level. This enables optimisation potential not utilised today. Secondly, the prototype shows a working, practical evaluation of the architecture across multiple layers (D.A.9 [7, Elastic NetInf Deployment], D.A.4 [8, Load-Adaptive Elastic NetInf Deployment]): NetInf is running in a virtual environment managed by ADT and deployed over the CloNe testbed. Both the architecture and the prototype lay the foundation of future research in topologically optimised application deployment.

4 Interaction between NetInf and OConS

The relationship between NetInf and OConS appears to be straightforward.

- From the perspective of OConS, a NetInf system is just a normal application. OConS is laying "under" the NetInf system and makes no use of its functions.
- From the perspective of NetInf, OConS is, essentially, just a packet transport mechanism. NetInf is, by virtue of the convergence layer concept, able to use packet transports of different capabilities. NetInf is also able to realise in such a convergence layer, albeit in a simplistic fashion, some functions that are required for the operation of NetInf but not offered by a particular packet transport.

One aspect that could reduce the benefit of the simple approach for the integration of OConS and NetInf is that the NetInf protocol messages need to travel hop by hop from the requester to the known location of the content (obtained either through name resolution or by forwarding the request up to the content provider) and back to the requester to allow or benefit from on path caching and request aggregation. We expect that in early NetInf deployment the routing decision in the NetInf layer will be based on the named object and information on the source of the object and not taking the network state in consideration. The OConS packet transport service would then be initiated and terminated on two adjacent NetInf nodes, leaving little room for OConS mechanisms to be fully profitable.

If the deployment of NetInf is done using the migration approach presented in D.A.4 [8], there may be multiple OConS nodes between two adjacent NetInf nodes. This would leave room for taking advantage of the OConS mechanisms. However, those benefits would be reduced and potentially disappear as the density of NetInf nodes in the network increases, reducing the "distance" between each of them.

At this point, interactions between the packet transport layer (OConS in the current case), the NetInf Convergence Layer and NetInf will require a higher level of integration. Information about the network state will have to be used by the routing and forwarding functions in NetInf CL and NetInf for increased efficiency of the network resources. Choice will have to be made on how to distribute specific functions in the stack. Using the example of the multipath transport, we will investigate the benefits and disadvantages of different option in the functional distribution between those elements.

Let's first recall the basic interfaces between OConS and NetInf from previous deliverables (D.B.3, D.C.2 [1, 3]).

4.1 Interfaces

4.1.1 NetInf Protocol Layer

The NetInf protocol layer performs routing and forwarding between NetInf nodes based on NDO names. NDO names are independent of the location of the object in the network topology.

NetInf protocol provides a number of message formats to communicate between different the nodes. These messages are created using a generic message format which is referred to as, an ICN message in NetInf. This generic format consist of the following format.

The abstract NetInf protocol that has been developed is based on the use of three pairs of messages and responses:

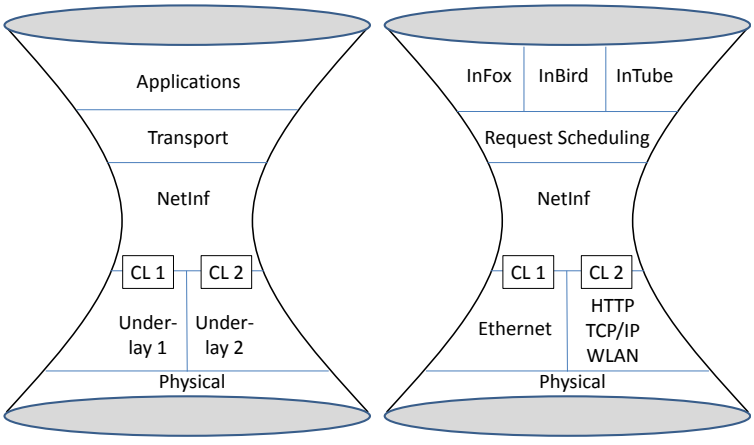
GET/GET-RESP The GET message is used to request an NDO from the NetInf network. A node responds to the GET message if it has an instance of the requested NDO; it sends a GET-RESP that uses the GET message's msg-id as its own identifier to link those two messages with each other.

PUBLISH/PUBLISH-RESP The PUBLISH message allows a node to push the name and, optionally, a copy of the object octets and/or object meta-data. Whether and when to push the object octets vs. meta-data is a topic for future work. Ignoring extensions, only a status code is expected in return.

SEARCH/SEARCH-RESP The SEARCH message allows the requestor to send a message containing search keywords. The response is either a status code or a multipart Multipurpose Internet Mail Extension (MIME) object containing a set of meta-data body parts, each of which MUST include a name for an NDO that is considered to match the query keywords.

The NetInf protocol layer relies on Convergence Layer (CL) to transport the messages between NetInf nodes. Many realisations of the CL has been realised using Hypertext Transport Protocol (HTTP) over Transmission Control Protocol (TCP), User Datagram Protocol (UDP) and the Disruption/Delay Tolerant Network (DTN) Bundle Protocol (BP) as transports for NetInf messages.

4.1.2 NetInf Convergence Layers



a) Schema b) Example

Figure 4.1: NetInf protocol stack, assuming a node with two convergence layers over two different underlays

The NetInf protocol specifies messages for node-to-node communication, their semantics, and corresponding node requirements. Different deployments will use different link layers and underlays — with a variation of services and properties. NetInf accommodates this via *CLs* that map the conceptual protocol to specific messages, transactions, or packet exchanges in an existing, concrete protocol. A CL provides framing and message integrity for NetInf requests and responses for

communication between two nodes as its main service, but specific CLs can provide additional services as depicted in Figure 4.1.

For example, NetInf-over-IP would require a CL that encapsulates, and potentially fragments and reassembles NetInf messages for transfer in IP packets and validates message integrity (e.g., by CRC). NetInf-over-Ethernet, on the other hand, could be done with a slim CL, but running over layer 2 would restrict the scope of services that a NetInf node could offer, so there are trade-offs when considering which CL(s) to use. Figure 4.1 depicts a schematic and sample NetInf stack for different CLs/underlays; it also shows that between applications and NetInf, additional functions like request scheduling (for implementing flow control, congestion control and other transport layer functions) can be inserted. With CLs, NetInf runs over quite different types of network links, including uni-directional or heavily delay-challenged links.

While specific CLs *can* provide transport protocol functions (reliability, flow control, congestion control), there is a need for transport layer functions across CL links. In NetInf, such functions are implemented on-top of the NetInf layer, e.g., by request scheduling/retransmission decisions.

D.B.3 [1] provides a detailed description of the NetInf protocol and specific CLs.

4.1.3 OConS Service Access Point

The main external interface with OConS is the Orchestration Service Access Point (OSAP). Applications such as NetInf communicate their requests regarding the desired connectivity services, to be set-up by the Service Orchestration Process (SOP), and receive notifications about the status of the requested connectivity services. The desired connectivity requirements are communicated by the application to OConS by the means of a demand profile.

Depending on the mechanism's specificity, the orchestration of an OConS service may span a single link, a group of links and nodes (network level), or affecting the complete end-to-end flow.

D.C.2 [3] provides more information on OConS OSAP.

4.2 Complementary Functions

At the top end of a NetInf convergence layer, a service access point for the NetInf protocol has to be realised (for details, refer to Section 2.2 of D.B.3 as the up-to-date version). At the bottom end, various transports can be used, e.g., UDP and HTTP. To have OConS and NetInf complement each other, all that is needed is to develop a corresponding convergence layer.

Such a CL could be simplistic, and use OConS as a simple packet transport. In a more clever CL, specific OConS capabilities can be exploited to better serve the NetInf protocol interactions required at the NetInf interface at the top of the CL. We have develop in SAIL, by way of example, one CL that leverages OConS's multi-path capabilities.

The OConS multi-path capable CL performs the task of identifying an appropriate forwarding strategy to retrieve the content required by the NetInf based applications. The CL uses chunk based request pipelining to request and retrieve the required content. It consist of a strategy selection and operation module that is able to control the flow of chunked content with an AIMD mechanism. By obtaining information related to the quality of the paths under the control of this CL, it is able to control how the requests for content are distributed to these different paths.

Further CL variations exploiting other OConS capabilities (e.g., in the wireless domain) are possible and not hard to develop. However, that would not really add too many scientific insights, and hence, we contended ourselves with this proof of concept.

In an extremely sophisticated version, it is conceivable to imagine that the OConS orchestration mechanism could become aware of the internal workings of a NetInf system. That could entail, for example, manipulating NetInf's name resolution tables from an OConS orchestration system when

better information about network proximity or congestion becomes available. But as discussed in Section 2.1, such a close integration has the high risk of close entanglement between two subsystems that can be neatly separated, all in return for benefits that are not at all clear. We hence decided not to pursue this option.

4.3 Commonalities

Areas of possible overlap are:

- Name resolution is likely to happen both in NetInf and in OConS. We do not perceive this as a problem as the name spaces are vastly different, they are working on different topologies, and the name resolution processes are very different. In any case, name resolution always happens at all layers of all communication stacks (provided the concept of "name" is correctly understood; even, e.g., IP "addresses" are, in the correct sense, names that are resolved in MAC addresses by the ARP protocol).
- NetInf, in absence of such a multi-path-on-top-of-OConS CL, might implement multi-path functionalities itself. While that might be the case, this is not necessarily a problem since that is (a) fairly easy to do in NetInf as the information-centric nature removes many of the multi-path complications, which a more generic OConS-based multi-path realisation has to deal with, (b) only an optional module in the NetInf core that is not optimised to exploit the underlying topology – unlike a corresponding OConS-based solution.

So there are indeed some limited areas of overlap, but these seem like a fair price to pay for independent deployability.

4.4 ICN Multisource and Multipath Communication

ICN is inherently offering multi-source communication: there is a one-to-many mapping of NDO names to object copy locations. NDOs can be published at several locations, and ICN's caching can create additional copies that can be considered as candidates for request forwarding decisions.

This potential plurality of object copies enables different strategies for ICN nodes:

parallel requests to different next hops: an ICN node can forward GET requests to multiple next hops in order to enhance reliability or to minimise latency.

load balancing: an ICN node can distribute a series of requests (perhaps for a series of NDOs) over multiple next hops using some scheduling approach such as Round-Robin in order to distribute load.

If *load balancing* can leverage path diversity, i.e., the different next-hops lead to request and response transmissions over disjoint path, a significant performance benefit can be achieved. Generally, a request scheduling and next-hop selection process has to decide which requests to forward over which paths, and (potentially) which flow-control regime to apply for that.

In ICN, the assumption for multi-source communication is normally that a certain application relies on requesting and delivering a series of Named Data Objects (NDOs). For example a video stream could consist of a set of NDOs, each of them representing an Application Data Unit ("chunk") – a fragment that had been created by the publisher.

A requestor would request each of these NDOs individually, and the network (i.e., network nodes) may decide how to leverage multiple options for next hops in order to achieve multi-path communication (i.e., forwarding requests for different NDOs over different next hops in parallel).

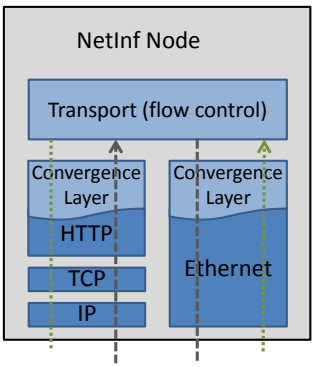


Figure 4.2: NetInf Transport protocol stack

In NetInf, there are essentially two alternatives for achieving multi-source and multi-path communication: 1) multi-source communication as a NetInf transport function, and 2) multi-source communication within a Convergence Layers (leveraging multi-source communication capabilities of an underlying network). We describe these two approaches in the following:

4.4.1 Multi-source Communication as a NetInf Transport Function

Figure 4.2 depicts the conceptual NetInf network stack architecture that is used in most NetInf implementations such as the NNRP router. A NetInf node uses one or multiple Convergence Layers and implements transport functions such as flow control on top of them, i.e., transport function logic is implemented in the NetInf message forwarding layer.

Figure 4.3 depicts a sample network setup where such a node could utilise multiple next-hop options (over different Convergence Layers) for sending/forwarding NDO GET requests. We have described the details of a transport protocol and its implementation for NNRP in D.B.3 [1].

The advantage of this approach is that it is a pure ICN approach – the transport function does not need to know anything about underlying networks. Convergence Layers take care of that. Forwarding and request scheduling decisions can be made based on forwarding information and a permanent performance assessment of the different next-hops.

Another advantage is that in some cases large information objects may be divided in smaller chunks and a master NDO is created to identify the set of NDOs corresponding to those chunks to ease the retrieval (e.g. HD movie split in smaller pieces). In that case the NetInf layer knows the relation between those smaller NDOs and makes the assumption that they could be retrieve from similar sources. A pipelining approach to the GET requests, or monitoring delays on multiple potential paths can help to define the load-balancing strategy.

4.4.2 Multi-path Operations using OConS

The operations of the OConS Multi-path NetInf mechanism is performed in a phased manner. These phases generally perform the tasks of setting up the mechanism in the required node and then performs the multi-path retrieval/delivery of content in NetInf.

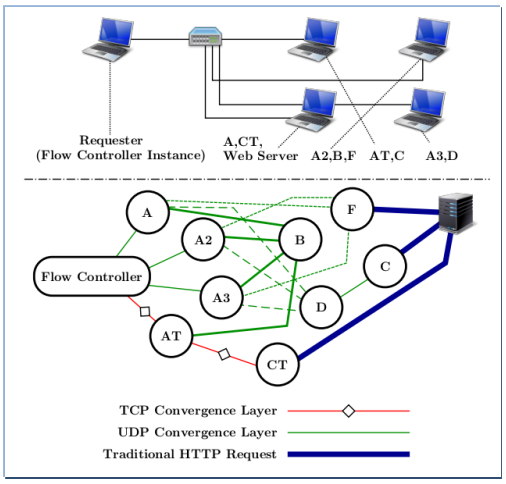


Figure 4.3: Sample NetInf multi-source network setup

4.4.2.1 Orchestration

The OConS framework consist of a number of different mechanisms. One of them is the Multi-path mechanism. The first phase of operations, known as Orchestration, is to identify the appropriate mechanisms and then setup these mechanisms in the different nodes. This activity involves a number of tasks that start with the discovery of the mechanism to finally identifying the different OConS framework elements to use.

In the case of the Multi-path mechanism the orchestration results in identifying the DEs to use that identify and operates the multi-path strategy, the EEs through which the decisions are implemented and the IEs from which to retrieve information to help in the decision making process.

The orchestration operates at the client nodes where decisions and the enforcement is done. The information to make decisions is received from sources in the client as well as from the network.

4.4.2.2 Operation of Mechanism

The operation of the Multi-path mechanism in NetInf involves the controlling of the multi-path capable CL to perform the retrieval of content using the multiple paths that the NetInf node has to the network. Figure 4.4 shows the messages that are passed between the different components in the client node when operating the multi-path mechanism (strategy).

The NetInf enabled application requests for a content providing the NI name. This is resolved by the NRS to a list of possible locations from which the content can be retrieved. This information is used by the OConS CL to retrieve the content NDO by NDO. The OConS CL consist of the application interface module, strategy module, ocons module and the number of MP UDP CL instances that are created to handle each of the paths.

The application interface module is responsible for generating the NetInf GET messages for the data chunks of the content required. The strategy module performs the selection of the MP UDP CLs to forward the GET messages. The strategy module evaluates the forwarding strategy to use regularly, triggered by the receipts of NDOs (chunks) and also based on other information provided by the OConS elements in the network.

The Multi-path mechanism assumes that the content is segmented (chunked) into smaller NDOs that could be retrieved one by one. The CL performs the task of identifying the segments (chunk)

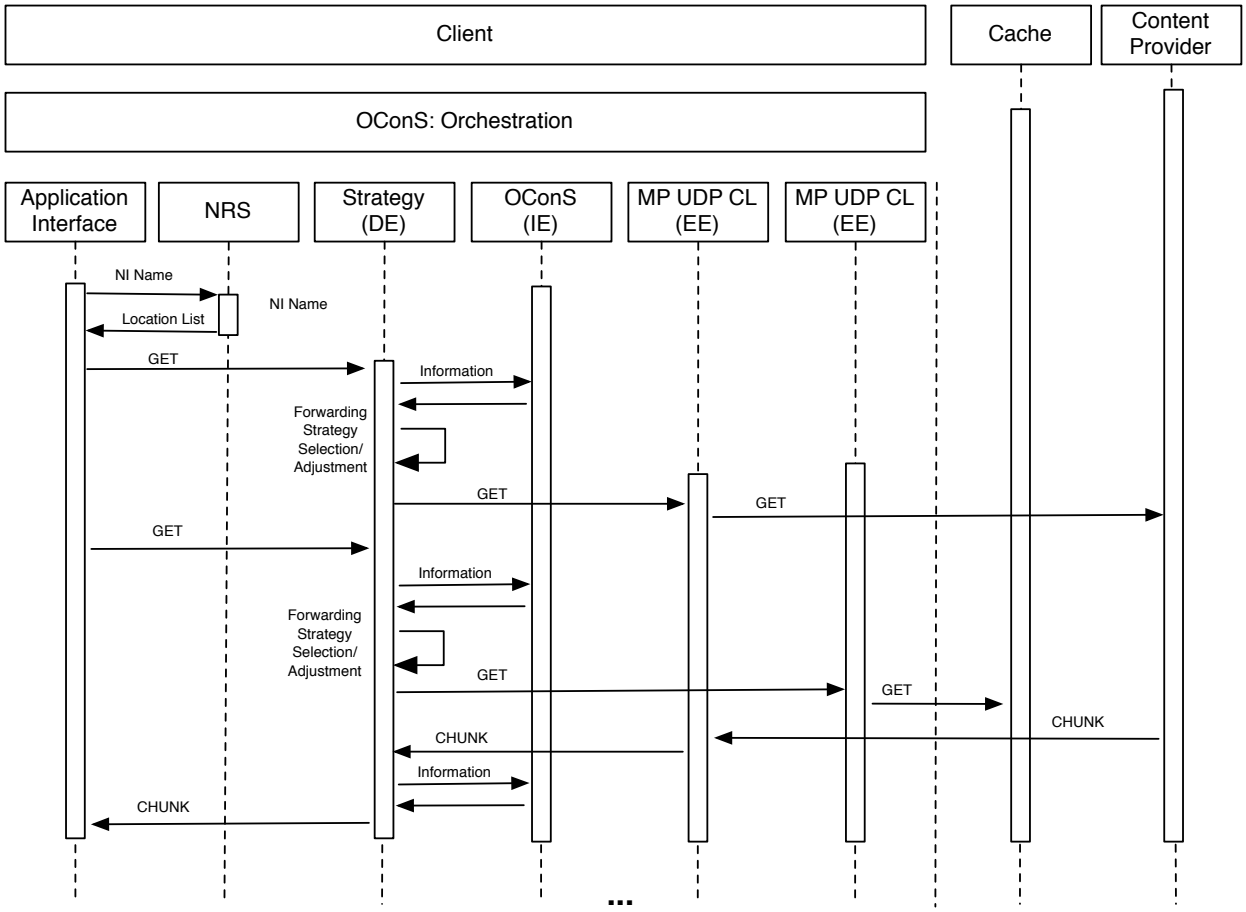


Figure 4.4: OCons Multi-path mechanism

to retrieve the NDOs, suffixing sequential numbers to the name of the original NDO. The retrieved segments are passed directly to the application in the right order but without doing reassembly.

4.5 Summary

ICN and NetInf supports multi-source communication natively. Any node with multiple interfaces can apply multi-source strategies, which is one of the significant advantages of ICN over other networking approaches. In NetInf, a node could implement different Convergence Layers that enable the usage of different interfaces, and a node could stripe sets of requests across all of these interfaces (or a sub set of them) – depending on forwarding information. Convergence Layers are not aware (and do not have to be aware) of NDO contexts and multi-source strategies.

The OConS approach introduces a slightly different perspective with the assumption that there is an underlying network fabric that provide multi-source communication. In NetInf, message transport over specific networks is implemented over Convergence Layers – consequently the OConS multi-source support would be implemented in a CL. This approach can be implemented, but is has the following disadvantages:

- CLs would need knowledge about different NDO locations. Normally this is a NetInf layer task – here OConS would have to instruct the CL.
- In order to benefit from multi-source communication within a CL, the CL implementation

would either have to split requests for one NDO to requests to a set of sub-NDOs (and receive fragments for those later) which opens questions for caching and name-data integrity.

- In a multihop NetInf setting, multi-source decision can only consider knowledge within a specific network segment (CL link).

As illustrated by the multi-path/multi-source example, the NetInf layer is the best place for making NDO forwarding decision as it has information about NDOs that the convergence layer does not have. However the forwarding decision mechanism can benefit of information obtained from the convergence layer and in particular from OConS.

5 Interaction between OConS and CloNe

The basic observations about the relationship of OConS and CloNe are:

- OConS manages how data flows from one *given* communication endpoint to another, possibly and typically choosing between different transport mechanisms when doing that (and collecting network status and context information to enable this decision) – this information selection and mechanism choice to connect two (or more) endpoints is called *connectivity service*. Typical choices to be made are use of single- or multi-path transport; selection between alternative paths; parameterisation of protocols. OConS can answer questions like: "given this specific end-point, what transport will give me $\leq$ 10ms latency?"

- CloNe, on the other hand, manages virtual networks within administrative domains and the interconnection of virtual networks across administrative domains, possibly by choosing between different virtual network management systems and interconnection technologies. The result is dynamic creation and management of a message forwarding *network service*, called *Flash Network Slice* (FNS).

  CloNe typically relies on the underlying network virtualisation technologies to optimise communication, but handles negotiation between domains over selection of interconnection mechanisms. CloNe may use information about networking capabilities to determine placement of virtual machines and storage.

These basic observations imply the following relationships:

- CloNe does have a notion of establishing connectivity end points, but it is connectivity between entire networks, specifically dealing with the problem of connecting them via administrative boundaries. The typical choice to make here is where to place an endpoint and, once it is placed, to add such an endpoint to the routing tables. CloNe does *not* worry about how the particular data flow between such endpoints – to this end, CloNe is widely agnostic about the particular connectivity service and use non-optimised IP/UDP/TCP flows just as well as optimised OConS connectivity services; it has no notion of "best possible" connectivity.

  Moreover, CloNe has a notion of resources on the level of virtual infrastructures and attempts to best decompose or partition such virtual infrastructures across administrative domains (dealing with the information hiding problem of competing operators).

- OConS, on the other hand, does not have a notion of a virtual network. It considers a given network topology, along with the given communication stacks, and tries to make optimal decisions on how to send packets between endpoints.

  OConS has a notion of resources on the level of individual paths, packet forwarding performance, etc.

5.1 Terminology

As CloNe and OConS had different perspectives on the problem at hand, they have used different terms for similar concepts. So the first step to be taken in identifying how OConS and CloNe

overlap or complement each other is to map the terminology used on both sides. We have picked the main terms used in both architecture and provided a mapping in Table 5.1. The OConS terms mainly refer to the data center interconnect use case (D.C.2 [3, Section 5.2] and D.C.4 [11, Section 3.1]) while the CloNe terms are from D.D.3 [2, Chapter 2].

OConS	CloNe	Remark
OConS domain	Intra-provider layer and Resource layer	
OConS provider	Infrastructure Provider	OConS domain provides mainly networking resources
Distributed Cloud Manager	Infrastructure service provider	Different names for the same concept.
Distributed Cloud Protocol	Distributed Control Plane	Same acronym, same concept, but with different names for historic reasons.
Domain Control Unit	Intra-provider resource manager	This role is not clearly identified in CloNe.
Domain Control Clients	-	No direct equivalent in CloNe, as CloNe did not studied the intra-provider internal architecture.
Orchestration Service Access Point	Infrastructure Provider interface (e.g OCNI)	OSAP is more generic than the Infrastructure Provider interface when it comes to connectivity requests as it encompasses many types of connectivity services. However, OSAP does not cover computing or storage resources management which are included in the CloNe interface.

Table 5.1: Terminology Mapping between OConS and CloNe.

5.2 Complementary Functions

In consequence, CloNe and OConS complement each other in several ways:

- In the simplest case, CloNe would simply use OConS connectivity service to establish a best-possible packet from between two endpoints – where CloNe had decided where these endpoints should be.

- In a more advanced complementation, one has to realise that CloNe does provide a virtual network: a network in which a communication stack can be recursively deployed. There is no conceptual boundary to using OConS inside this virtual topology and use its orchestration mechanisms.

  In a full-scale deployment, that would mean there are two (or even more) OConS instances working independently of each other. We hence see that this defeats the purpose of orches-

tration across layers. Such an uncoordinated use of separate, virtualised Open Connectivity Servicess is, hence, not recommendable.

- The full combined power of
  - CloNe-based network virtualisation and
  - OConS-based orchestration of information collection and mechanism selection and parameterisation

  could come to bear if the OConS instances inside the virtual networks could orchestrate their operation with OConS instances outside the virtualisation. But while this approach has considerable theoretic elegance, it goes considerably beyond the current scope of SAIL and seems to require much further research to work out details. Still, the overall architectural approach of SAIL could support it.

5.3 Commonalities

There are some areas where there is indeed some overlap:

- Selecting a particular transport between two given endpoints: In a sense, this is the core function of OConS, and it invests considerable effort in this choice. CloNe, on the other hand, also has to make this selection in absence of a deployed OConS system, but it will then fall back on very simplistic considerations (essentially, establishing an UDP flow between these endpoints without even attempting to optimise).

- This is specifically highlighted in the data center interconnect use case. In the case of CloNe, we are attempting to demonstrate that we can construct virtual infrastructures in different data centers managed by different infrastructure service providers and connect them up through a wide area network operated by a third infrastructure provider, making it appear to be a single virtual infrastructure. The connectivity model between these centers is rather simple: the Single Router Abstraction (SRA) with a very simplified Network Management System (NMS) that can control different flavours of *off-the-shelf* connectivity[1]), not really amenable to sophisticated network optimisation, but suitable for the virtualisation discussion.

  In the case of OConS, the emphasis is on the efficient interconnectivity of two data centers; the "single infrastructure" image is not relevant for that investigation and not addressed. In particular, the load of the network is monitored and, using a centralised controller for decisions, the allocated network resources are adapted to the actual load, possibly rerouting the traffic when more bandwidth is needed. This is far from feasible in the CloNe inter-data-center connectivity model.

5.4 Coordinating Open Connectivity Services and Cloud Networking

The main task of CloNe is to deal with the high-level connectivity definition between data centers and the deployment of the VMs in them. This high-level connectivity includes the SRA in order to hide specific details of the network such as topology.

On the other hand OConS deals basically with the details of the connectivity layer in order to provide a service detailed setup (i.e. multi-path, QoS, etc.). OConS also covers the runtime control of the connectivity making specific network layer tune in order to comply with the service quality parameters (QoS/QoE) and providing restrictions.

[1] An example is the testbed implemented in WP-D which happens to use Multi-protocol Label Switching (MPLS) and OpenFlow-based network segments

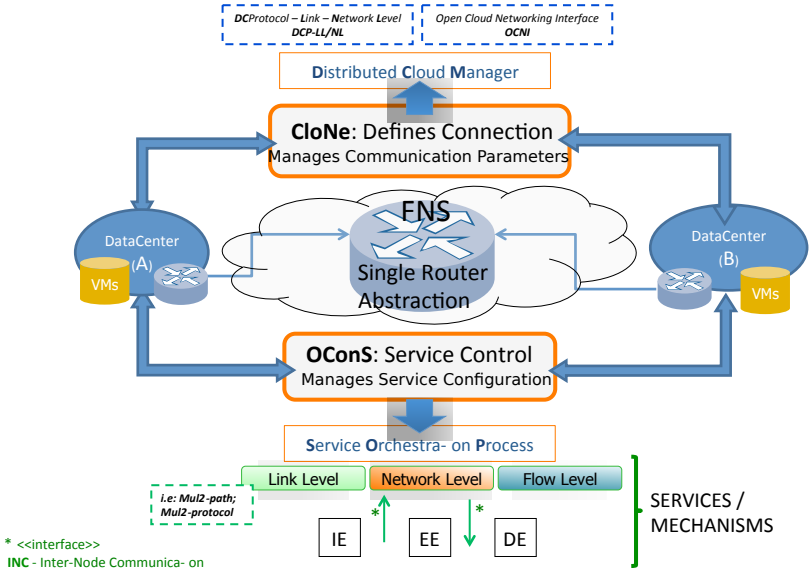


Figure 5.1: Relationship between OConS and CloNe

Figure 5.1 shows the relationship between OConS and CloNe in the context of the data center use case. The key idea is that CloNe deals with the resource configuration on the data centers and establishes the connection between them, and OConS manages the data center interconnection in real-time.

Each WP has their own interfaces in order to obtain service information from the network, DCs or the service itself and order actions (Multi-path configuration, Quality of Service (QoS) parameter modification, etc.) to orchestrate a service, in this case, creation of a Flash Network Slice of DC resources and providing the Single Router Abstraction.

In the case of CloNe, there are different interfaces used, most relevant being:

- Distributed Control Plane (DCP): Includes the Link Negotiation Protocol (LNP) for the link level and the Cloud Message Brokering Service (CMBS) for the network level. It binds the data centers and the wide area network together allowing for true automated cross-domain deployment of virtual infrastructures.

- Infrastructure Service Interface (ISI): Used to convey the network requirements (following the SRA) to a domain. It follows the Open Cloud Networking Interface (OCNI) proposal and has been showcased using the pyOCNI implementation.

These interfaces are used to exchange messages with the network about service failure, lack of resources, etc. The messages from CloNe are forwarded through the ISI to the service managers in OConS (IE, EE, DE) in order to trigger events and actions on the service control plane.

In OConS, the SOP is capable of orchestrating an OConS service composed by one or several OConS mechanisms. OConS users communicate with the SOP by means of the Orchestration Service Access Point (OSAP). Through the OSAP, users communicate their requests regarding the desired connectivity services, to be set-up by SOP, and receive notifications about the status of the requested connectivity services. In order to store the data of the various mechanisms, rules and policies, as well as the network state, the SOP is connected to a database, known as Orchestration Register (OR).

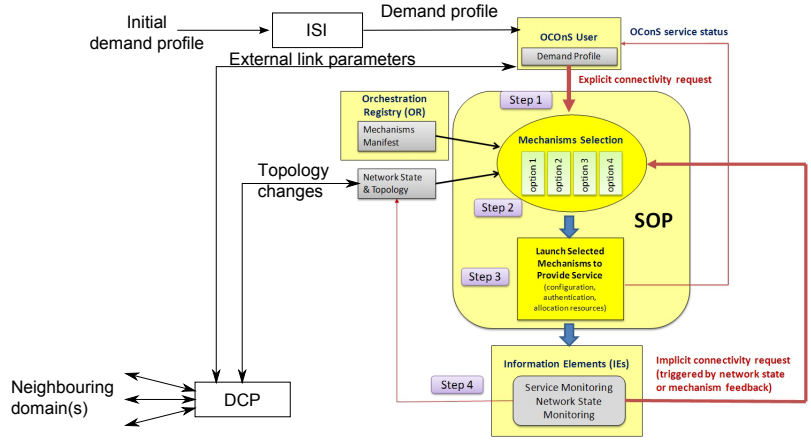


Figure 5.2: A simple integration of the CloNe interfaces with the OConS Service Orchestration Process

In order to interconnect the OConS process with the CloNe ones a specific interface is needed. CloNe uses the Infrastructure Service Interface to control the resources and the information provided by this interface and to inform other entities external to CloNe. So the ISI and specifically the DCP protocols (LNP and CMBS) should be used to inform OConS entities, such as the SOP and the OR, in order to orchestrate the whole service.

The Infrastructure Service Interface (ISI) is a cloud networking extension to the Open Cloud Computing Interface (OCCI) core model to add support for networking technologies other than Virtual LANs (VLANs). It also exposes a service model for network resources similar to that offered by the Infrastructure as a Service (IaaS) model. Based on HTTP, it is a platform independent and extension friendly protocol. It is used both in the data center and in the network operator areas and provides a rich data model and operations that facilitate the management of the network, compute and storage resources.

A specific interface is needed to exchange and negotiate technical parameters in a domain-to-domain specific manner. The protocol used to exchange control information needed by administrative purposes is collectively called Distributed Control Plane (DCP) and includes the CMBS and the LNP. CMBS is a service for exchanging messages between the domains/providers in CloNe. It exchanges information between providers and enables not native messaging functions (like publish/subscribe, broadcasting, etc.). LNP is used for negotiating connectivity related parameters needed to connect, create and manage distributed virtual infrastructure.

For example, to create and establish a Virtual Private Network (VPN) among different providers, an OCNI request for a VPN link is received on the Resource Manager (RM). Once this message is acknowledged, the LNP is initiated with both end-points as specified on the request. After the LNP is completed, the RM delegates to the NMS, who is in charge of creating the VPN.

Integrating CloNe with the SOP in OConS

Figure 5.2 shows a proposal for the integration of the CloNe interfaces and the SOP developed in OConS. At setup time, the initial topology and service profile of a specific domain are received at the ISI. The process extracts the demand profile that is fed into the SOP as an explicit connectivity request. This allows OConS to establish the connectivity in the OConS domain. In addition, DCP-LNP provides complementary requirements for stitching the internal connections with the

neighbouring domains.

Listing 5.1: Requesting an OpenFlow path over an OConS enabled infrastructure using pyOCNI

```
{
    "kind": {
        "term": "CloNeLink",
        "scheme": "http://schemas.ogf.org/occi/ocni",
        "class": "kind"
    },
    "occi.core.id": "OConS_OF_Link1",
    "occi.core.title": "OConS_OF_Link1",
    "occi.core.summary": "CloNeLink with openflow mixin",
    "mixins": [
        {
            "term": "OConSOpenFlowLink",
            "scheme": "http://schemas.ogf.org/occi/ocni",
            "class": "mixin"
        }
    ],
    "links": [],
    "attributes": {
        "ocni.clonelink.availability":[
            {
                "ocni.availability.start":"08:00",
                "ocni.availability.end":"12:30"
            },
            {
                "ocni.availability.start":"14:00",
                "ocni.availability.end":"18:00"
            }
        ],
        "ocni.clonelink.state":"active",
        "ocni.clonelink.bandwidth":"100Mbps",
        "ocni.clonelink.latency":"100ms",
        "ocni.clonelink.jitter":"",
        "ocni.clonelink.loss":"0.01%",
        "ocni.clonelink.routing_scheme":"unicast",
        "ocni.OConSOpenFlowLink.IPv4_src":"192.168.10.4",
        "ocni.OConSOpenFlowLink.IPv4_dst":"192.168.10.8",
        "ocni.OConSOpenFlowLink.IPv4_proto":["tcp","icmp"]
    }
}
```

Listing 5.1 shows how a request through the ISI looks like. The `ocni.clonelink` elements contain the demand profile and the `ocni.OConSOpenFlowLink` elements define the end points of the path for a given domain as well as the protocols, that will be allowed on the link. These elements could (and, in further developments, will) be integrated in the `ocni.clonelink` structure. [2]

During the lifetime of the service, the SOP may decide to change the connectivity endpoints in order to maintain the Quality of Service or Quality of Experience level. These topology changes are fed to the Distributed Control Plane, which communicates them to the affected neighbouring domains.

Complementary information on the integration of CloNe and OConS can also be found in D.C.4 [11, Section 3.1].

[2] However, they were kept in an isolated structure to highlight the fact that OpenFlow is being used in this specific case, due to the novelty of this technology and the fact that it is very much in the spotlight.

6 Themes

6.1 Security

SAIL wants to avoid that security is taken as an afterthought. However, the WP specific research topics need their individual solutions and R&D emphasis. The common approach to the projects activities in cloud networking, wireless access and information centric networks is hence that WP individual security objectives were identified and individual solution were considered and elaborated. In accordance with this approach, the project wide coordination concerning the cross cutting technical theme security has the objectives [6]:

- to create a coherent and comprehensive security framework in which the project results can be evaluated regarding common security objectives and coherent security solutions, and
- to understand what potential misuse could be and what remaining deficiencies are.

As such, the project aimed to achieve that security solutions in the individual WPs are free of contradictions and duplication of work. A gap analysis, to identifying missing functions to complement the given security objectives are presented below. As security and privacy-ensuring solutions may sometime create trade offs with usability, the cross WP activities created awareness for these dependencies.

As security specific architectural guidelines were released the addressed ambitions can be summarised as: avoid security retrofit; elicit right-size requirements; obey provided guidelines and best practice; minimise component technology and avoid unnecessary dependencies.

The following section will present an overview on the security specific topics addressed in cloud networking, wireless access and information centric networks and provide a kind of architectural security "map" enabling to reader to understand the researched topics and leading quickly to further detailed documentation. This is complemented with further information that can be understood as open R&D questions that are "for further study".

6.1.1 Security Goals And Security Specific Research Results

6.1.1.1 NetInf

The key security issues we have addressed in NetInf are those directly tied to naming and name data integrity, i.e. *Naming Security* and *Content Integrity*. *Naming Security* refers to services that provide cryptographic strength binding between an object name, and the form of the object returned by the ICN in response to a request. *Content Integrity* in NetInf is related to Naming Security and provides cryptographic assurance that the returned object value is unmodified since some previous event such as object creation or publication. These mechanisms has been well defined for NetInf and has resulted in the ICN naming scheme 'ni', which standardise how to use hashes to name information objects has been approved to become an IETF Standards track RFC. For details please see: http:// tools.ietf.org/html/draft-farrell-decade-ni.

Privacy and legal intercept: ICN can support fully anonymous communication as NDOs need not be tied to any publisher or node, new private keys for signing can be generated on the fly. Alternatively ICN can offer a full-blown Orwellian communication system if ICN would require that

Figure 6.1: Security Aware Design And Development Process [12]

only NDOs signed by approved and registered keys are allowed in the network. Thus regulation is needed to ensure a reasonable level of privacy at the same time as it is possible to provide legal intercept. Possible approaches/techniques includes control of resolution services or a requirement to register your keys before before being allowed to publish. The NetInf technology that we have designed offers full freedom in this respect. The next step will have to be taken on the regulatory arena.

6.1.1.2 OConS

The innovation oriented approach taken with OConS, which aims at higher flexibility and more openness in control of emerging and existing (transport and others) mechanisms, requires a thorough security consideration. This is a usual demand for new technology, regardless if it follows in a clean slate approach or in an migration oriented approach as in the case of OConS. In both cases the proposed technology will encompass its own security properties. In the migrational approach there may be security relevant effects on the existing environment e.g. imposing new threats or enabling new side channels. It is therefore only consequent if OConS is made subject of an threat analysis, which is a usual entry into a security aware design and development process (Figure 6.1). It is early enough to avoid security retrofit because technology decision have not matured. It is late enough to not unnecessarily influence the core functionality that distinguishes OConS.

The main topics OConS addresses are the lack of coordination among mechanisms on various level that have evolved over long time and covey data in out network infrastructures. Additionally, OConS addresses the network control lacking flexibility and expressiveness to improve handling (often domain specific) policies, which govern e.g. the mobile core network, firewalls, fixed networks, forwarding rules for switches [3]. Here, OConS enables to combine, orchestrate and thus integrate different mechanisms to improve.

The envisaged security objectives of network architectures that encompass OConS deployments is however not different from what we know and define today as protection goals to be achieved: ensure availability and integrity while maximise the misuse protection. Overall, the security should not weaker than with existing architectures, however, it maybe even better is achievable for economically viable costs [3, Sec. 6.3 and Annex F].

As such OConS deployments may have side effects due to the integration in existing infrastructures or the proposed technology per se may encompass security flaws, a threat analysis was indicated [3, Annex F]. As approach for this analysis and due to the stage of development of the OConS architectural framework an attacker oriented threat analysis was proposed, considering that

an attacker could obtain some control over the orchestration process or a given mechanism by:

- Impersonating a legitimate OConS node,
- Postponing or replaying OConS messages or,
- Modifying or forging messages.

This exercise was made to understand the attack surface and potential vulnerabilities and to guide further developments. As one immediate consequence the protection of the Intra/Inter-Node Communication (INC) has been addressed [3, Annex C.4]

6.1.1.3 CloNe

Schoo et al. [13] presented a comprehensive and structured security analysis of the Cloud Networking infrastructure, and described the security challenges, as considered in SAIL, in order to ensure the legitimate usage of cloud networking resources and to prevent misuse. Out of the security challenges covered in the paper, the major security challenges were identified as information security, virtualisation management, isolation, misuse protection, and identity management. These security challenges are further elaborated [2, Section 2.6]. The security activities in WPD focus on designing security solutions which address these security challenges.

The security solutions designed as a part of the CloNe security architecture include:

Access control In order to address the security challenges raised by information security and virtualisation management an access control policy function was elaborated [2, Section 4.5.5], [10, Section 4.6]. The access control policy function is responsible for determining whether an infrastructure service user is allowed access to a particular resource. The access control policy function utilises an authorisation logic, which enables distributed authorisation across multiple operator administrative domains, and allows fine-grained control of virtual resources managed by the administrator of the physical resource set.

Auditing and Assurance function In order to address the security challenges raised by isolation and virtualisation management an auditing and assurance function has been defined [2, Section 4.5.3]. It comprises of two major sub-functions, namely, a TPM based auditing function and a CloudAudit [14] based assurance function. The TPM based auditing function attests the geographic location of the physical resources provisioned to the infrastructure service user [10, Section 6.5]. The CloudAudit based assurance function is responsible for verifying that the deployed resources match the security requirements entered by the infrastructure service user.

Security goal translation function In order to to translate the security requirements provided by the infrastructure service user, and generate concrete resource specifications which can be deployed on the underlying resource set, the CloNe security architecture utilises the security goal translation function [2, Section 4.5.2].

SIEM based Intrusion Detection System In order to address the security challenges raised by improper misuse protection, a Security event and Information Management (SIEM) based Intrusion Detection System (IDS) has been described [2, Section 4.5.1]. The SIEM based IDS function detects security incidents which occur in the CloNe infrastructure, and provides a real-time analysis of these security alerts.

Identity Management In order to address the security challenges raised by identity management, an identity management framework has been presented [2, Section 4.5.4]. It is capable of identity provisioning, creating and maintaining access control policies, and authentication of legitimate infrastructure service users.

6.1.2 Security Perspectives Concerning Harmonisation Of Results

Due to the nature of the different R&D topics in cloud networking, wireless access and information centric networks that request individual security specific solution e.g. because of different design goals or optimisation requirements, all the topics identified have their own right to exist. Overlap of duplications could not be identified, although on a higher level of abstraction, topics were named similar. For example, naming and addressing follows very different and R&D topics specific requirements in NetInf and CloNe when taking a more details look. Briefly outlined, the optimisation of NetInf on naming and addressing follows the efficient distribution of data where as naming and addressing in CloNe is optimised to enable interoperability of distributed data centers.

From WPC perspective and assuming that OConS takes a linking role in an overall architecture, then secure and reliable interoperability and service discovery with NetInf and CloNe needs investigation; potentially new threats of these combinations need to be identified due to interoperable combination of different security policy domains.

6.1.3 Outline of Further Security Related Research

For **NetInf** there are still a number of areas that remain largely untouched and offers fine opportunities for future studies. These include *Content Authentication* which can be related to naming security and provides assurance (possibly cryptographic) as to the identity (via some identifier(s)) of entities associated with the object - perhaps an "owner" or the entity that published the object. *Content Confidentiality* which is a service that ensures that the plain text value or an object is only visible to authorised entities. There may be different forms of this service with different "end-points", corresponding to the usual end-to-end vs. hop-by- hop security services offered in host-based networking. *Authorisation* refers to the general ability of an entity to control which other entities are supposed to be able to gain access to an object. *Provenance* is a service that allows authorised entities to trace the overall actions of an ICN upon a set of objects. Finally we have *Copyright/IPR issues in the context of caching*. The caching of objects have implications on how the legal framework deals with caching objects, in transit, that would mean copyright and/or other legal violations if they were regarded as being in your possession.

The **OConS** work, i.e. concept, architecture definition, tentative architectural framework, etc., has been started in SAIL and an initial threat analysis has been completed. This is in accordance with a typical development process as outlined (cf. Section 6.1.1.2). According to such a development process the next steps should be [3, Sec. 6.3] to revisit security goals or objectives, formulate security requirements and design those functionality to become intrinsic functions of OConS, which iteratively has to be validated again [12].

Concerning **CloNe** the security solutions described in Section 6.1.1.3 should be evaluated by deploying and integrating them into a CloNe test bed. This would help validate their efficacy in addressing the CloNe security challenges [2, Section 2.6]. The validation of security solutions can be initiated by evaluating the performance overhead and accuracy of the access control function and the auditing and assurance function. Validation of the access control function would evaluate the maximum detail level at which the access control policies can be specified, the accuracy of the authorisation logic in translating infrastructure service user requirements into access control policies, and the performance overhead associated with the deployment of the access control policies. Validation of the assurance and auditing function would evaluate the auditing mechanisms, which

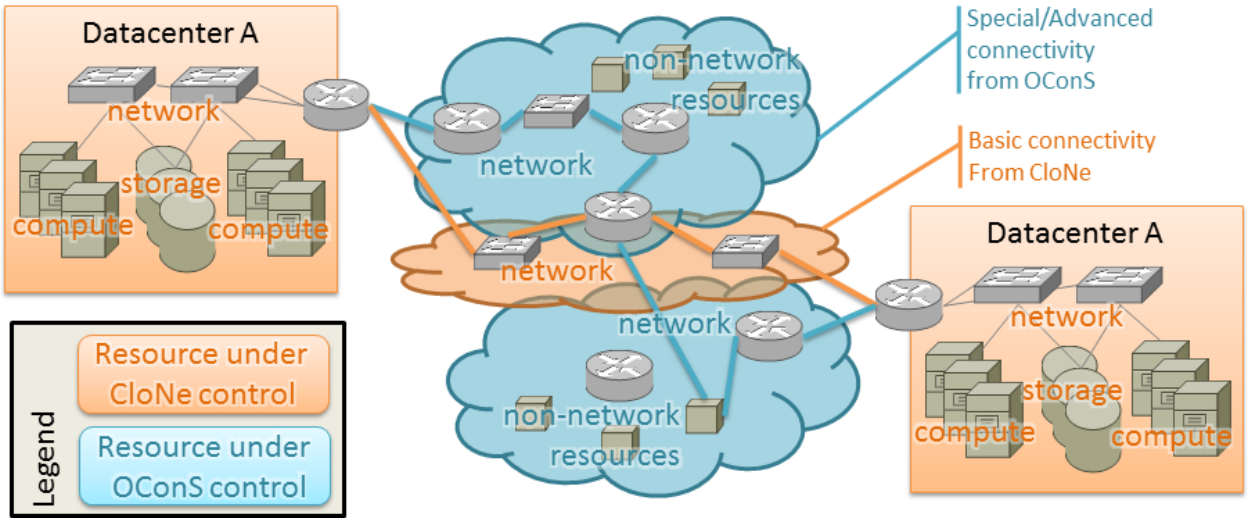


Figure 6.2: Distribution of control between CloNe and OConS. Inside Datacenters CloNe is in charge of resource control. Between Datacenters two options exist: Either (i) the special connectivity services of OConS are used, or (ii) CloNe network operators provide and manage basic connectivity.

are used to ensure that the security related resource specifications are reliably deployed on the set of resources, and the performance overhead associated with the auditing process.

6.2 Network Management

NetInf, CloNe, and OConS all require some control over network resources. The goal of the network management theme is to ensure that coexistence of different mechanisms is possible. Looking at management aspects across all of the technical work packages, specifically, Deliverable D.A.2 (Architecture Guidelines and Principles) and Deliverable D.A.9 (Description of overall prototyping use cases, scenarios and integration points) yielded some possible overlap in terms of who is responsible for which ressources.

6.2.1 Resource management overlap

This overlap exists mainly between OConS and CloNe, both providing significant requirements and corresponding functions for managing resources. As a first step towards the resolution of the resource management tasks between both work packages, the Network management Theme has proposed that both work packages follow a more collaborative resource management approach, rather than a strictly segregated one. The results of this effort are visible in Chapter 5 (particularly in Sections 5.3 and 5.4).

Figure 6.2 shows the final distribution of control between CloNe and OConS. Inside Datacenters CloNe is in charge of resource control. Between Datacenters two options exist: Either (i) the special connectivity services of OConS are used, or (ii) CloNe network operators provide and manage basic connectivity. Thus OConS and CloNe collaboratively manage only limited aspects of a resource. While this approach replicates some functionality in both management domains, it also enables CloNe and OConS to be deployed in isolation without losing functionality.

6.2.2 Mutual benefit from sharing information

The inter-WP prototypes (as reported in D.A.9) demonstrate some examples of cross-WP network management. The Elastic NetInf use-case for example outlined the requirement to monitor network utilization from both the CloNe managed hardware and the NetInf controlled software overlay. This use-case can be extended with e.g. OConS-based inter-site connectivity services. For the mentioned prototype the proposed solution is to add a management module in between NetInf and CloNe. From an architecture perspective this module is located in the CloNe domain. Although this module needs to be application aware, it requires tighter integration with CloNe's resource allocation and management. This application awareness can be provided by allowing an application developer (in this case the NetInf operator) to design a resource scheduling algorithm for CloNe.

6.3 Inter-Provider Theme: Domain definition in SAIL

The three technical work packages in SAIL have worked on complementing aspects of the relation between providers. The approach taken by each of them has been independent and orthogonal issues has been addressed. In order to identify the requirements, SAIL uses single domain providers.

6.3.1 Network of Information

There are several notions of domain in NetInf which are linked but not identical to the domains of OConS and the Flash Network Slices (FNSs) of CloNe.

The first notion pertains to the interaction with the legacy Internet which is seen as an external object or domain interacting with NetInf both at the naming level and at the transport level.

A second notion linked to the previous one is the domain of domain names which is taken into account in the ni naming scheme through the optional Universal Resource Identifier (URI) authority field.

Finally the main notion of domain and thus inter-domain is the one corresponding to current partition of the Internet routing fabric, each partition being administered by a single commercial entity.

Inter-domain in the context of today's Border Gateway Protocol (BGP-4) Autonomous Systems (ASes). The Internet connects independent networks which are interconnected. Each domain within Internet corresponds to one zone under responsibility of one network operator, in a broad sense, today's solutions to the inter-domain problematic are really inter-AS solutions through BGP-4 use.

NetInf may use two mechanisms concurrently, name resolution and name-based routing. Name resolution is supported by at least one Name Resolution System (NRS) which answers with a list of locators corresponding to the name of the object. So each operator or AS may need to have at least one NRS.

6.3.2 Open Connectivity Services

OConS tries to optimise the multi-path/multi-layer/multi-domain transport and routing, to efficiently exploit networking resources heterogeneity, to maintain the Quality of Experience (QoE) in highly mobile situations, to support challenged networks through self-* mechanisms, and to effectively and autonomously control the data-centre (inter)connectivity. A Domain in OConS is:

- a portion of the network implementing a given set of OConS mechanisms
- a set of nodes/entities under an administrative body that own/operate them
- a set of nodes/entities under certain "policies"
- a set of nodes/entities in a "trust" relationship

OConS use-cases focus on edge-to-edge communication of (large) data-centres across the core network. The OConS user in this case is a complete network who creates a different demand for connectivity between such (business-, provider-) users and the OConS provider or operator of a sub-network/domain.

Multi-domain heterogeneous technologies coexist where convenient edge-to-edge interoperation and connectivity across the core networks and the requirements for communications are dynamically changing (e.g., real-time services).

We follow a model based on a centralised control server entity per domain (as a Decision Making Entity; here called Domain Control Unit (DCU)) connected to client's switching or routing entities called Domain Control Client (DCC) within the domain for monitoring traffic and collecting information by an Information Management Entity (IME); and manipulating forwarding and routing tables in the switch/routing entity (Execution and Enforcement Entity).

Distributed/collaborative architecture to discover features and available services is needed. Also resilience, scalability and manageability, can be applied to any OConS functionality. From a routing perspective, mechanisms must address advanced mobility management, security and multi-path to enable efficient edge-to-edge transport and optimised services. The edges may also be defined by a set of multi-domain end points including a number of multiple administrative domains, policy domains, trust domains, etc.

6.3.3 Cloud Networking

CloNe relies on the concept of *Flash Network Slice* (FNS) to establish virtual network infrastructure for the cloud users. The FNS allows to connect computing and storage resources distributed in the network in a single or multiple domains. To establish those FNSs, CloNe architecture is modular and can provide its own mechanisms but can also rely on the services offered by OConS such as the wide area network (WAN) Interconnectivity for Virtual Networks. One issue identified in FNS management is how to identify the end-points that might be in different domains.

Today, the most widespread form of virtual networking across service provider domains are Layer 3 VPNs, to deploy a multi-domain VPN, require a prior agreement between providers about VPN-specific network parameters.

To make this process dynamic and automatic, an inter-domain control plane between network providers must be put in place. In the CloNe architecture, the set of protocols and interfaces used to implement coordinating actions across administrative domains is known as DCP. Figure 6.3 shows how the DCP cooperates with the domain NMS and provides the inter-domain communication with other CloNe domains.

A Flash Network Slice can be seen as an advanced form of virtual network by incorporating requirements such as on-demand provisioning and resource elasticity, which have largely resulted from the need to support the Cloud Computing paradigm. A FNS spans several network domains, this means that an inter-provider control plane is required.

6.3.4 Conclusions and Further Work

In future networks, the amount and semantical richness of the information that needs to be exchanged between domains will surpass the limits of what can be done with the Border Gateway Protocol. Additionally, the separation between data forwarding plane and control plane will impose restrictions on the way the network is partitioned. SAIL has explored north-bound and horizontal interfaces in the control plane to exchange setup requirements for virtualised environments that offer virtualised storage computing and networking resources from a technical point of view. Additionally, the possibility of creating an OConS service to actually control the virtualised infrastructure shows that further work on a consolidated inter-domain architecture is needed. This

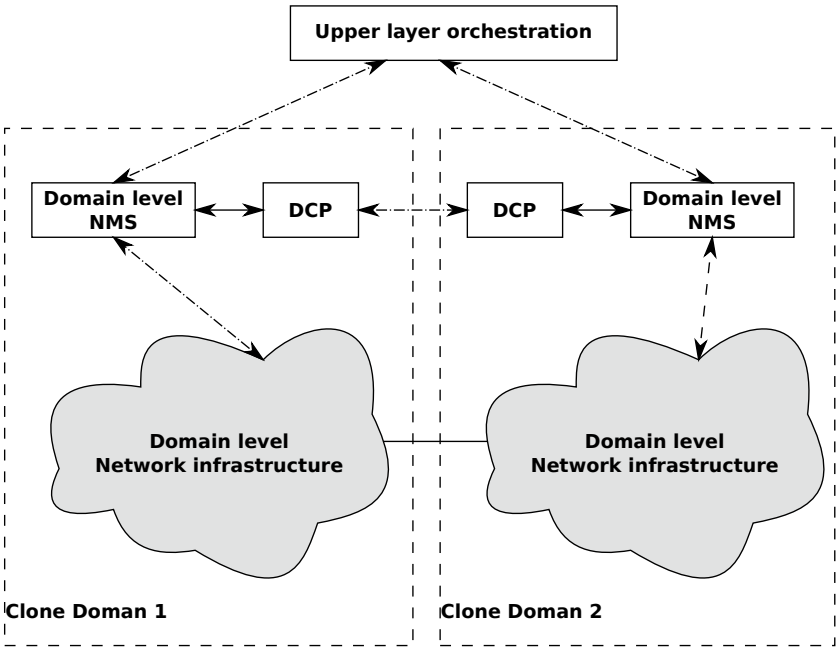


Figure 6.3: CloNe domains and the Distributed Control Plane (DCP)

work has started with an Internet draft in the Internet Research Task Force (IRTF) [15, 16] and will be continued beyond the duration of SAIL in the Open Networking Foundation (ONF) and the Virtual Networking Function (VnF) study group at ETSI.

7 Conclusion

We have provided an architecture for future networks that is built around Content, Connectivity and Cloud. The selected approach based on interaction building blocks allows for parallel development of the different technologies. It identifies and specifies a reduced number of interactions between the building blocks limiting the dependencies both during the development as well as at deployment time. This enables gradual deployment of the different SAIL components as the proposed technologies evolve and mature.

The interface and interactions between the components have been studied from a theoretical point of view but they are also the result of experimentation and prototypes. This ended up in a better understanding of the interactions and interfaces.

CloNe proposes a flexible framework for providing elasticity of application deployment. We have clarified the interactions over the ISI that enable elasticity of the deployed applications. We have also described how NetInf leverage the CloNe elastic deployment to achieve a friendly migration and be able to adapt to the varying demand. While CloNe can react relatively quickly (depending on the actual metrics and thresholds) to changes in the demand, it is still to be studied to which extent NetInf can fully benefit of this short turn around time or if it requires longer life-cycle to take into account the resource required for starting VMs and populating caches. However, even in the longer life-cycle, the CloNe elasticity mechanism can automate deployment and ease the management of the NetInf system.

OConS provides the mechanic for implementing the FNS proposed by CloNe, adding the tools for supporting elasticity at the network level in coordination with CloNe. Additional features that could have been provided is the support from OConS for selecting resource placement based on the information available from the network: topology, load, ... Without this function from OConS, CloNe can still rely on coarse grain information available (domain location and topology) but may end-up with sub-optimal solutions.

Using multi-path/multi-source forwarding, we have illustrated different options for the integration of NetInf and OConS. We observed that a much tighter integration between NetInf and OConS is required for NetInf to fully benefit of the information available in OConS. This is due mainly to the fact that routing and forwarding decisions is moved up the stack from the packet layer to the NetInf layer. The selection of locators, routing hints or request and response paths could be improved by the network knowledge detain by OConS. However, this would highly increase the dependency of NetInf over OConS, which was not deemed desirable as we progressed in SAIL.

In addition to the interfaces between the main components, we have addressed three Themes that were relevant to the three work packages: Security, Inter-provider and Network management. For those Themes, we have ensured that no duplication or contradiction in the solutions were put forward by the work packages.

List of Acronyms

ADT Application Deployment Toolkit

AP Application Provider

AS Autonomous System

BGP-4 Border Gateway Protocol

BP Bundle Protocol

CL Convergence Layer

CloNe Cloud Networking

CMBS Cloud Message Brokering Service

DC data center

DCC Domain Control Client

DCM Distributed Cloud Manager

DCP Distributed Control Plane

DCU Domain Control Unit Distributed Hash Table

DTN Disruption/Delay Tolerant Network

ETSI European Technical Standards Institute

FNS Flash Network Slice

HTTP Hypertext Transport Protocol

IaaS Infrastructure as a Service

ICN Information Centric Networking

IME Information Management Entity

IRTF Internet Research Task Force

ISI Infrastructure Service Interface

LNP Link Negotiation Protocol

MIME Multipurpose Internet Mail Extension

MPLS Multi-protocol Label Switching

NDO Named Data Object

NetInf	Network of Information
NRS	Name Resolution System
OCCI	Open Cloud Computing Interface
OCNI	Open Cloud Networking Interface
OConS	Open Connectivity Services
OF	OpenFlow
ONF	Open Networking Foundation
OPI	Online Performance Indicator
OR	Orchestration Register
OSAP	Orchestration Service Access Point
pyOCNI	Python Open Cloud Networking Interface
QoE	Quality of Experience
QoS	Quality of Service
RM	Resource Manager
SAIL	Scalable Adaptive Internet Solutions
SOP	Service Orchestration Process
SRA	Single Router Abstraction
TC	Trusted Computing
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
URI	Universal Resource Identifier
VLAN	Virtual LAN
VM	Virtual Machine
VnF	Virtual Networking Function
VPN	Virtual Private Network
WAN	wide area network
WP	work package

List of Figures

List of Tables

Bibliography

[1] The SAIL Consortium. Final NetInf Architecture. Deliverable FP7-ICT-2009-5-257448-SAIL/D.B.3, SAIL project, November 2012. Available online from `http://www.sail-project.eu`.

[2] The SAIL Consortium. Refined CloNe Architecture. Deliverable FP7-ICT-2009-5-257448-SAIL/D.D.3, SAIL project, October 2012. Will be available online from `http://www.sail-project.eu`.

[3] The SAIL Consortium. Architectures and mechanisms for connectivity services. Deliverable FP7-ICT-2009-5-257448-SAIL/D.C.2, SAIL project, February 2013. Will be available online from `http://www.sail-project.eu`.

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

[5] Bengt Ahlgren, Pedro Aranda, Prosper Chemouil, Luis M. Correia, Holger Karl, Sara Oueslati, Michael Sllner, and Annikki Welin. Content, connectivity, and cloud: Ingredients for the network of the future. *IEEE COmmunications Magazine*, July 2011.

[6] The SAIL Consortium. Draft architectural guidelines and principles. Deliverable FP7-ICT-2009-5-257448-SAIL/D.A.2, SAIL project, July 2012. Available online from `http://www.sail-project.eu`.

[7] The SAIL Consortium. Description of overall prototyping use cases, scenarios and integration points. Deliverable FP7-ICT-2009-5-257448-SAIL/D.A.9, SAIL project, December 2012. Available online from `http://www.sail-project.eu`.

[8] The SAIL Consortium. Final migration description. Deliverable FP7-ICT-2009-5-257448-SAIL/D.A.4, SAIL project, February 2013. Will be available online from `http://www.sail-project.eu`.

[9] Margaret Chiosi et al. Network Functions Virtualisation Introductory White Paper. White paper, Group of Network Operators, October 2012. Available online from `http://www.tid.es/es/Documents/NFV_White_PaperV2.pdf`.

[10] The SAIL Consortium. CloNe - Description of Implemented Prototype. Deliverable FP7-ICT-2009-5-257448-SAIL/D.D.2, SAIL project, July 2012. Will be available online from `http://www.sail-project.eu`.

[11] The SAIL Consortium. Applications for connectivity services and evaluation. Deliverable FP7-ICT-2009-5-257448-SAIL/D.C.4, SAIL project, February 2013. Available online from `http://www.sail-project.eu`.

[12] Charles B. Haley, Jonathan D. Moffett, Robin Laney, and Bashar Nuseibeh. A Framework for Security Requirements Engineering. In *SESS '06: Proceedings of the 2006 international workshop on Software engineering for secure systems*, pages 35–42, New York, NY, USA, 2006. ACM Press.

[13] Peter Schoo, Volker Fusenig, Victor Souza, Márcio Melo, Paul Murray, Hervé Debar, Houssem Medhioub, and Djamal Zeghlache. Challenges for cloud networking security. In *Proceedings of the MON-AMI Conference*, Santander, Spain, Sept 2010.

[14] C. Hoff, S. Johnston, G. Reese, and B. Sapiro. CloudAudit 1.0 - Automated Audit, Assertion, Assessment, and Assurance API (A6), July 2010.

[15] Haiyong Xie, Tina Tsou, Diego López, Ron Sidi, Hongtao Yin, and Pedro Andrés Aranda Gutiérrez. SDNi: A Message Exchange Protocol for Software Defined Networks across Multiple Domains. `http://tools.ietf.org/id/draft-yin-sdn-sdni-00.txt`, Jun 2012.

[16] Haiyong Xie, Tina Tsou, Diego López, Ron Sidi, Hongtao Yin, and Pedro Andrés Aranda Gutiérrez. Software-defined networking efforts debuted at ietf 84. *IETF Journal*, Oct 2012.