Objective FP7-ICT-2009-5-257448/D-4.5

Future Networks

Project 257448

"SAIL – Scalable and Adaptable Internet Solutions"

D-4.5
(D.C.5) Demonstrator for Connectivity Services

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-4.5
Document Title:	**(D.C.5) Demonstrator for Connectivity Services**
Document Responsible:	Lucian Suciu (FT-Orange)
Document Editor:	Michael Soellner (ALUD)
Authors:	Ramón Agüero (UC), Pedro A. Aranda Gutiérrez (Telefónica I+D), Luis Fco. Díez (UC) Marta Garcia (UC), Olivier Mehani (NICTA), Susana Perez (Tecnalia), Horst Roessler (ALUD), Peter Schefczik (ALUD), Michael Soellner (ALUD), Lucian Suciu (FT-Orange), Asanga Udugama (UHB)
Target Dissemination Level:	PU
Status of the Document:	Initial Version
Version:	1.0

Production Properties:

Reviewers:	Luis M. Correia (IST-TUL), Fabian Schneider (NEC), Benoit Tremblay (EAB)

Document History:

Revision	Date	Issued by	Description
1.0	2013-02-28	Michael Soellner	Initial Version

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 is a public deliverable of the EU-FP7 project SAIL (Scalable Adaptive Internet Solutions) and reports the final realization of prototype components and the demonstrator activities for the SAIL work package 'Open Connectivity Service (OConS)' at project end. This comprises prototype realizations of OConS architectural concepts such as orchestration or abstract IEs and EEs for multiple mechanisms. Moreover, demonstration details of cross-WP cooperation are given from an OConS perspective.

Keywords:

open connectivity services, prototyping, demonstration, cloud networking, network of information, openflow, dtn, icn, multi-path, distributed connectivity control, distributed mobility management

Executive Summary

This document is a public deliverable of the EU-FP7 project SAIL Scalable Adaptive Internet Solutions [1] and reports the state of the final prototype realizations and demonstrator activities for the SAIL work package 'Open Connectivity Service (OConS)' at project end.

It focuses not only at the WP-specific parts, but includes also use cases and scenarios including details of cross-WP cooperation (with work package (WP) 'Cloud Networking (CloNe)' and with WP 'Network of Information (NetInf)') from an OConS perspective based on the project-wide description in the deliverable D.A.9 "Description of overall prototyping use cases, scenarios and integration points" [2].

The practical evaluation work in WP OConS started with the design and realization of a first experimental project phase which resulted in intermediate demonstration show cases. These results were presented and discussed at a project-internal workshop in January 2012. Further plans towards more integrated solutions have been described in the deliverable D.C.3 "Demonstrator Specification and Integration Plan" [3].

Based on these initial components and the gained experience, this document now extends D.C.3 and describes the further steps of OConS prototype and demonstration activities which focus in a second phase on the updated use case scenarios (from D.C.1-Addendum [4]), the work package internal cooperation based on the progress of the OConS architectural framework (specified in D.C.2 [5]), and the cross-WP cooperation with CloNe and NetInf.

Selected steps for prototyping OConS architectural concepts and realization of reusable components such as orchestration or abstract Information Management Entities (IEs) and Execution and Enforcement Entities (EEs) for common mechanisms are given here. For the OConS-CloNe interaction, we focus on developing an "elastic networking" use case scenario, whereas the OConS-NetInf interaction deals with multi-path connectivity services for Information-Centric Networks (ICN)-like content delivery, as well as improving Delay Tolerant Network (DTN) connectivity and routing.

Most of the prototype components have been successfully presented and demonstrated live at public events, such as the "Future Network and Mobile Summit (FuNeMS2012)", Berlin, July 2012, the "4th International Conference on Mobile Networks and Management (MONAMI'13)", Hamburg, September 2012, and the final workshop on "Future Media Distribution using Information Centric Networks", Stockholm, February 2013.

Contents

1 Introduction

This document is a public deliverable of the EU-FP7 project SAIL Scalable Adaptive Internet Solutions [1] and reports the state of the final prototype realizations and demonstrator activities for the SAIL work package (WP) 'Open Connectivity Service (OConS)' at project end.

It focuses not only at the WP-specific parts, but also includes use cases and scenarios including details of cross-WP cooperation (with WP 'Cloud Networking (CloNe)' and with WP 'Network of Information (NetInf)') from an OConS perspective, based on the project-wide description in the deliverable D.A.9 "Description of overall prototyping use cases, scenarios and integration points" [2].

The practical evaluation work in OConS started with the design and realization of a first experimental project phase which resulted in intermediate demonstration show cases. These results were presented and discussed at a project-internal workshop in January 2012. Further plans towards more integrated solutions have been described in deliverable D.C.3 "Demonstrator Specification and Integration Plan" [3].

Based on these initial components and the gained experience, this document now extends D.C.3 and describes the further steps of OConS prototype and demonstration activities which focus in a second phase on the updated use case scenarios (from D.C.1-Addendum [4]), the work package internal cooperation based on the progress of the OConS architectural framework (specified in D.C.2 [5]), and the cross-WP cooperation with CloNe and NetInf.

Chapter 2 recaps the framework for the prototyping and demonstration activities, as it was introduced, on a project-wide basis, in D.A.2 [6], and the requirements from a OConS point of view, as discussed in D.C.3 [3].

In Chapter 3 we describe the realization of reusable components of the OConS architectural framework. As a demonstration, the use case of orchestrating two different mechanisms is also presented. In particular, we assess the feasibility of OConS to smartly combine (orchestrate), an enhanced access selection mechanism with the execution of tunnelling management for distributed mobility support.

For the OConS-CloNe interaction, we focus on the development of an "elastic networking" use case scenario described in details in Chapter 4.

The OConS-NetInf interaction deals with multi-path connectivity services for Information-Centric Networks (ICN)-like content delivery, as well as improving Delay Tolerant Network (DTN) connectivity and routing, both of them are covered in Chapter 5.

Most of the prototype components have been successfully presented and demonstrated live at public events, such as the "Future Network and Mobile Summit (FuNeMS2012)", Berlin, July 2012, the "4th International Conference on Mobile Networks and Management (MONAMI'13)", Hamburg, September 2012, as well as during the final workshop on "Future Media Distribution using Information Centric Networks", Stockholm, February 2013. The explaining poster material used at these events is added in the Annex A of this document.

2 Approach and Methodology

In this chapter we revise the overall Scalable Adaptive Internet Solutions (SAIL) approach for prototyping, further elaborating the particular aspects which are relevant to OConS activities. It also discusses the main principles for validation tasks, and introduce the three main aspects which were addressed during the project lifetime, matching them with the architectural concepts and the use cases which have streamlined the OConS work. Finally, the chapter discusses the most relevant platform and integration choices which were made to carry out the various implementation tasks.

2.1 SAIL Approach for Prototyping

The goal of the experimental approach in SAIL has been to prove that the SAIL research results are mature innovations satisfying the initially postulated expectations, including:

- to demonstrate innovative (i.e differentiating) solutions (functions, features, scenarios) to known/existing requirements or problems, or
- to demonstrate basic (unique, 'first') solutions (functions, features, scenarios) to innovative requirements or problems.

Credible evidence for these innovations should be achieved by practical validation in multiple stages:

- By **prototyping**, used to validate specific aspects (properties) of innovative model, system or component designs against the initial assumptions (requirements). A prototype typically includes just a few aspects of the features of the eventual technical system. Basic prototype categories are, among others, proof-of-concept prototypes (checking feasibility), functional prototypes (checking required functions), or visual prototypes (checking look-and-feel).
- By **experimentation**, as a means in the scientific methodology that arbitrates between competing models or hypotheses. Experimentation is also used to test existing theories or new hypotheses in order to support or disprove them. However, an experiment may also test a specific aspect or previous results. From that, it is clear that experimentation requires a realization of a prototype.
- By **demonstration**, which is a scientific/technical experiment carried out for the purposes of proving scientific/technical effects or solutions, rather than for hypothesis testing or knowledge gathering (although it may originally have been carried out for these purposes).

It is worth recalling that SAIL, in general, and the OConS WP, in particular, did not aim at conducting field trials or massive test deployments in operating network environments.

For further considerations see Chapter 3.5 in D.A.2 [6].

2.2 Validation Principles

The stages as described above require not only a prototype realization of the innovative solution, but also a validation framework that is used to control the test environment and to allow identifying

initial - as reproducible as possible - conditions, including:

- Setting the environmental conditions and control the pre-conditions: their identification and control are needed in order to allow a reproducible or comparable course of experiments or demonstrations.
- Variation of input / modification of [scenario, network, component] configuration: In order to study the impact of different configuration parameters on the system behaviour, it shall be possible to modify those set-up conditions.
- Measurement and visualization of output / [network, component] behaviour: The validation framework shall be able to measure, monitor and visualize relevant system reactions and outputs. This is necessary in order to enable comparison between expected and actual system behaviour, as well as to conduct error debugging processes in case of failure.

The following topics describe the procedure to be followed for a best practice experimental design, and thus serve as guidelines on the validation process:

- Problem statement, identify the problem to be solved, the hypothesis to be tested, and/or the aspect/property to be validated
- Describe and model assumptions and abstractions, constraints
- Identify system pre-conditions and input parameters
- Specify expected experimental outcome
- Run prototype test, experiment or demonstration
- Document actual experimental outcome
- Evaluation, comparing the actual outcome with the expected one, considering as well the particular input parameters.

2.3 OConS Validation Aspects

The practical evaluation work in WP OConS started with the design and realization of a first experimental project phase, driven by early demonstration and experimentation activities of the partners accompanying the OConS technical framework definition, which resulted in intermediate demonstration show cases. These results were presented at the project-internal workshop in January 2012 (project month 18).

Based on these components and the gained experience, a further step of OConS prototype and demonstration activities have been defined in D.C.3 [3], which focuses, in a second phase (project month 19-30), on the updated use case scenarios (from D.C.1-Addendum [4]), the workpackage internal cooperation based on the progress of the OConS architectural framework, and the cross-WP cooperation with CloNe and NetInf.

D.C.2 [5] gives an overview about the novel OConS architectural framework and lists the mechanisms which, either separately or in combination, provide new functionality. OConS work in SAIL comprised the architectural framework, the actual functional architecture and the dedicated mechanisms at flow, network and/or link level. As part of the overall SAIL architecture, OConS needs to be able to cooperate with CloNe and NetInf across external interfaces.

As OConS is designed as an 'open system' with extensible architectural framework, it does not make sense to provide a single demonstrator as 'the' OConS prototype. It is also obvious that not all of the mechanisms introduced in [5] can be turned into a demonstration. Instead, we identified the critical parts of the new OConS approach and went for exemplary realization of the key aspects.

This way, the OConS final prototyping and demonstration scenario selected from the overall OConS portfolio comprises (see also Figure 2.1):

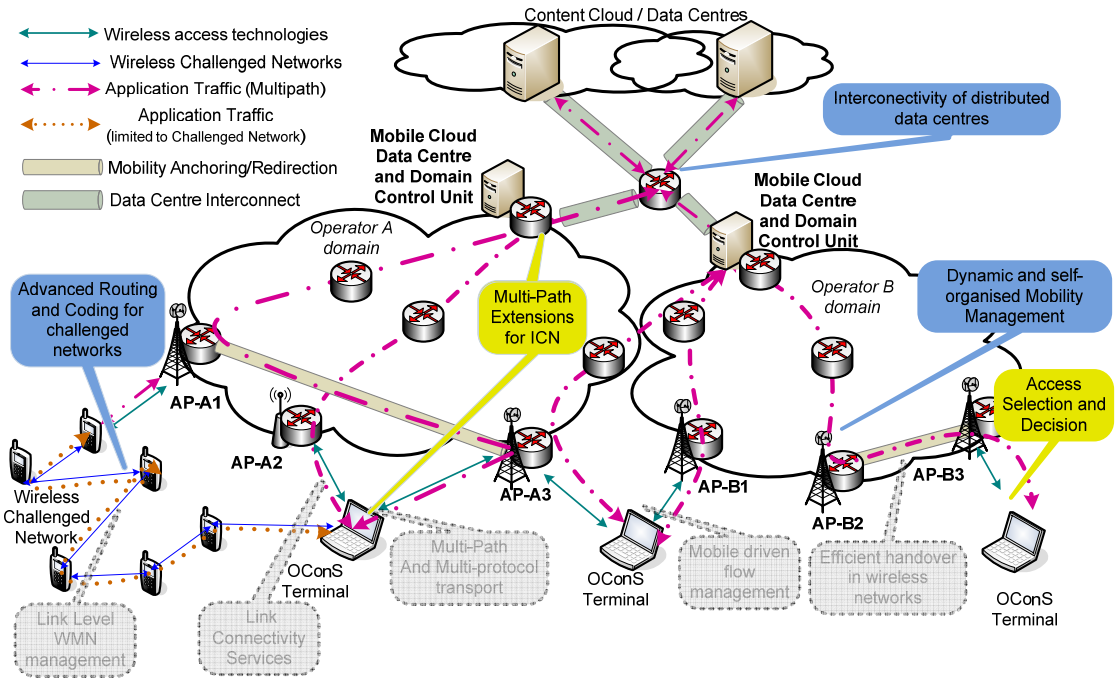


Figure 2.1: Selected OConS Architectural Components and Mechanisms for Prototyping and Demonstrations

- Chapter 3 describes the realizations of reusable components from the OConS architectural framework, such as the orchestration function and generic Information Management Entity (IE) and Execution and Enforcement Entity (EE) modules. As a demonstration, a use case in which two different mechanisms were orchestrated is selected. In particular we assess the feasibility of OConS to orchestrate an enhanced access selection mechanism with the execution of tunnelling management for distributed mobility support. This is an example in the area of multiple access networks.
- For the OConS-CloNe interaction, we focus on developing an "elastic networking" use case scenario, described in detail in Chapter 4. It uses network level mechanisms for creating virtual networks, representing datacentre instances, and the distributed datacentre interconnection over Wide Area Networks (WANs) by optimally selected and maintained flows, in close relation to the Distributed Cloud Manager (DCM) of CloNe.
- The OConS-NetInf interaction deals with multi-path connectivity services and their related link and forwarding mechanisms for ICN-like content delivery; it also addresses the improvement of DTN connectivity and routing. Both prototyping activities are reported in Chapter 5.

The main questions to be answered by the OConS prototyping and demonstration activities were:

- Is the OConS architecture framework open and flexible enough to cover many existing and up-coming connectivity mechanisms?
- Can the OConS architecture framework be realized at reasonable complexity (feasibility)?
- Can the OConS architecture framework complement reasonably/efficiently the concepts of CloNe?
- Can the OConS architecture framework complement reasonably/efficiently the concepts of NetInf?
- Can the function of the OConS architecture framework be demonstrated in a way to support and promote the developed new concepts?

- How can the OConS prototype realization contribute to the project-wide 'flash crowd' scenario?
- Which aspects/issues of the OConS architecture framework can be identified that need further practical/experimental exploration in the future?

Deployment on larger test-beds or scalability studies were out of scope of the OConS prototyping and demonstration activities.

2.4 OConS Platform and System Integration Choices

The guidelines for platform and system choices for the OConS prototyping and demonstration activities have been already discussed in D.C.3, Section 5.4.

These guidelines should enable a basic communication between involved OConS components in the demonstration set-up. Contents of messages are specified where necessary between the cooperating partners.

This allows for a pragmatic approach of cooperation between the involved components, as the focus of the OConS prototyping and demonstration activities is on the functional level, rather than at the level of standards-relevant bit-exact interface specifications.

According to the OConS architecture, an OConS system is a heterogeneous distributed system with components that run concurrently in different processes, either on the same node or on different nodes. For the sake of the project, OConS components are realized as processes in the same virtual machine or in different machines under the same host system. As development platform for access and infrastructure nodes, open platforms such as Linux and Android have been chosen. As an *open* system, by design, OConS does not have a dependency on a specific platform, as long as the components comply with the protocols and interfaces at the connecting reference points. OConS entities exchange messages in the form of requests and responses with each other, as well as through asynchronous notifications that are subscribed beforehand.

The main goal of the OConS prototyping and demonstration activities was to complement the theoretical framework with practical experience in a proof-of-concept manner. So their value is more providing the playground for experiments rather than to come up with a complete single demonstrator integrating all aspects of the OConS framework.

In the following chapters, we also discuss the specific platforms selected and the realization details in each of the scenarios and use cases, i.e., on end-terminals, on access nodes, and on core and datacentres nodes.

3 Architecture and Orchestration: OConS Framework Reusable Components

The OConS framework [5] has been conceived so as to enable a smart, yet quick, combination of connectivity mechanisms to be able to offer the end-user an overall better performance. One of the main ideas behind this objective is to ease the deployment of new services, taking advantage from the set of already deployed mechanisms. In this sense, a developer does not need to have a monolithic solution per scenario, but would be able to reuse those mechanisms which are already available.

The aforementioned goal is definitively a challenging one, since the number of mechanisms (for each of the identified levels) is remarkably high (see [7]). In order to assess the feasibility of this approach, we pursued a prototyping activity, in which we implemented the supporting functionality of the different OConS entities, including the signalling protocol. In this sense, it is quite straightforward to deploy a new entity (e.g. an IE), since the developer does not need to be worried about the specifics of the interaction with the rest of the OConS framework, but only about what the specific mechanism needs to particularly carry out (e.g. gathering the required information). The whole implementation task which is described hereinafter had this goal from its very beginning.

Once the barebones of the OConS functionality were available, the following step was to assess their feasibility. For that, we selected a couple of illustrative mechanisms: enhanced access selection and dynamic mobility management (see [5, 7]). With the former we demonstrated how the OConS framework can be used so as to dynamically modify (enhance) the operation of an already deployed mechanism (by the notification of the appearance of new relevant entities). Besides, we used the two of them so as to show the potential of the orchestration process. These two particular mechanisms were originally orthogonal of each other, but the OConS framework successfully enabled their combination, without major changes it their particular implementation details.

The potential of the OConS approach was shown in an demonstration, see section 3.2. In addition, some of the developed modules were designed so as to enable their quick reusability, so they might be exploited to deploy new mechanisms, which in addition, benefit from the OConS supporting functionalities.

3.1 Functional overview

3.1.1 Generic OConS Protocol Library

One of the aspects which are required in order to ensure the appropriate operation of the OConS framework is the inter-entity communication. It is worth recalling that OConS can be actually seen as a distributed set of components for which connectivity might not be fully defined and which can vary widely (according to, for example, the forwarding scheme in use); in this sense it becomes of utter relevance to offer a communication facility which properly decouples OConS messages exchange from the underlying communication technology. As described in [5], this task is performed by the Inter/Intra-Node Communication (INC), which can be described as a hub in

charge of relaying the messages to their corresponding destination, either locally or remotely. This entity thus appears as the cornerstone of the OConS communication.

In order to cope with the communication requirements which are present in every OConS node, a generic OConS protocol library has been developed; it implements the INC functionality as well as the OConS interfaces. The library has been fully implemented in $C/C++$ programming language. The most important functionalities provided by this library are described below, following a top-down approach:

- INC: an INC daemon has been implemented; it is able to distinguish OConS messages coming from either local entities or remote ones. Besides, it forwards the messages to the appropriate entity: local, by the corresponding Inter Process Communication (IPC) system (namely *Unix Domain Sockets*); or remote, by means of a suitable interface. In order to perform this operations, the INC stores the global OConS ID of an entity (i.e. the combination of the OConS node ID and the entity local ID, as described in [5]) and maps it onto the appropriate interface. Besides, due to the two communication levels which are considered in OConS (i.e. inter-entity and inter-node) the INC also has the possibility to piggy-back, encapsulating various messages from different entities towards the same node. In the OConS terminology, one or more OConS messages are encapsulated into the same OConS packet.
- Message handling: either the INC or the different entities must be able to handle the OConS messages and packets. The developed library also provides a set of helpers so as to ease the processes of message content manipulation and creation. In particular, they offer a simplified way to create and read OConS headers, encapsulate messages into OConS packets (piggy-backing) and encapsulate data (following the corresponding Type Length Value (TLV) format) into OConS messages.
- TLVs handling: finally, and as the abstraction at the lowest level, the library provides data formatting features. In this sense, it supports the creation and reading of TLVs in a generic manner (taking advantage of generic programming techniques) so as to facilitate the data abstraction and avoiding an always undesirable and error-prone bit-wise task.

In summary, the generic OConS protocol library offers a set of communication facilities which are the first step for further implementations of OConS mechanisms and services. It can be therefore used to ease the development of a particular entity, offering the means to appropriately communicate with other functional elements.

3.1.2 OConS Orchestration Functionalities

The main goal of OConS is to provide an open framework to support any type of connectivity service. This requires a rather high level of abstraction in defining its constituent components. In order to manage this intrinsic openness and versatility, three elementary and generic entities (namely IE, Decision Making Entity (DE) and EE) have been defined as well as the generic interfaces to communicate between them.

In order to support dynamic configuration, combination and instantiation of mechanisms and services, the instances of these three basic components need to be coordinated; this coordination is the main role of the *orchestration functions*.

Based on the previously described library, some basic orchestration functionalities have been developed to establish a basic operation of a connectivity mechanism. The testbed in which these functionalities have been implemented and tested is thoroughly described in [8] and it will be presented afterwards in section 3.2. The orchestration functionalities follow the description of [5]

and are provided as software modules able to carry out various orchestration tasks:

Entities registration (and their capabilities) in a local Orchestration Register (OR). By sending a **REGISTER_req** message, the entities are able to register their capabilities and acquire the needed IDs, both OConS node and entity IDs. We have also implemented a registration functionality which enables the OR to keep track of all the entities it is aware of, with operations such as the registration of a new entity (storing both IDs and capabilities) and the update of entity capabilities;

Mechanism verification. This happens once the entities register with the Service Orchestration Process (SOP). As capabilities from different entities are collected, the SOP is able to check whether a new mechanism is available (its building entities are already registered) or might be updated. This latter case means that a running mechanism could be improved after new entities which might improve its current operation have become available;

Remote entities discovery. In order to provide the abstraction the OConS framework requires (local and remote entities shall be managed alike) a remote discovery functionality has been also integrated, to maintain updated registers in the different ORs;

Configuration of entities by the SOP. Whenever a mechanism is to be initiated or updated, the SOP configures the mechanism DE by sending the corresponding **CONFIGURE_req** to enable its communication with the rest of entities involved in the mechanism (both local and remote);

Lower configuration level. A secondary storage facility is integrated within the DE, to keep the IDs of the entities involved in the mechanism, as well as a quasi-autonomous procedure to configure the corresponding IEs and EEs. Upon receiving the **CONFIGURE_req** message from the SOP, and before answering, the DE stores the information required to enable the communication with the rest of entities and sends configuration messages to each of them. This last procedure can be customized so as to carry out the appropriate configuration information, as will be further depicted in Section 3.1.3.

In summary, starting from the generic OConS protocol library, we have developed a set of orchestration functionalities which allow going a step beyond the implementation and orchestration of particular OConS mechanisms. These orchestration procedures are mechanism-independent and can thus be used as a starting point for the development of the OR, the SOP or other OConS entities.

3.1.3 Template for Generic OConS Elements

In addition to the particular functions that IEs and EEs can carry out, they follow rather similar patterns, specially in terms of design. Conversely, the DEs are characterized by a much more specific behaviour, which makes a general implementation quite complicated. In this section we present the generic implementation (templates) which have been designed and built for the IE and EE, using the communication and orchestration facilities which were described earlier.

In order to tackle such a generic implementation, we have made use of the *Template Method* behavioural design pattern (deeply described in [9]), which perfectly fits the purpose of this implementation. This library is implemented in $C++$ programming language and offers template classes for both IEs and EEs. The basic structure for both *templated* IE and EE objects is depicted in 3.1 and 3.2, respectively. In both cases, the classes implement a basic structure to deal with the different communication needs, using the aforementioned generic protocol library. Besides, each entity can be further customized so as to perform a particular action by means of the template parame-

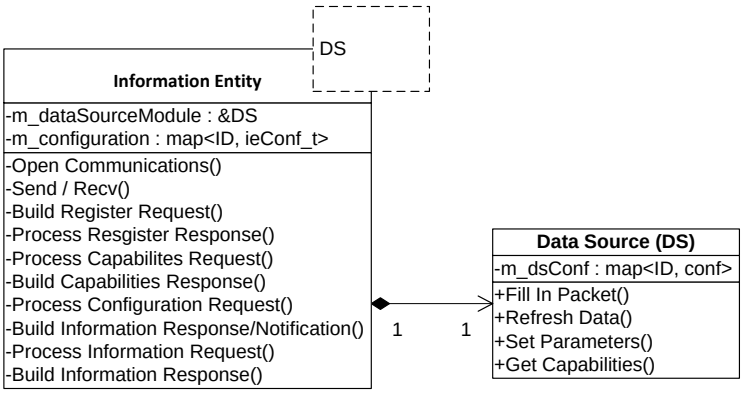


Figure 3.1: Structure of the generic IE object provided by the OConS library

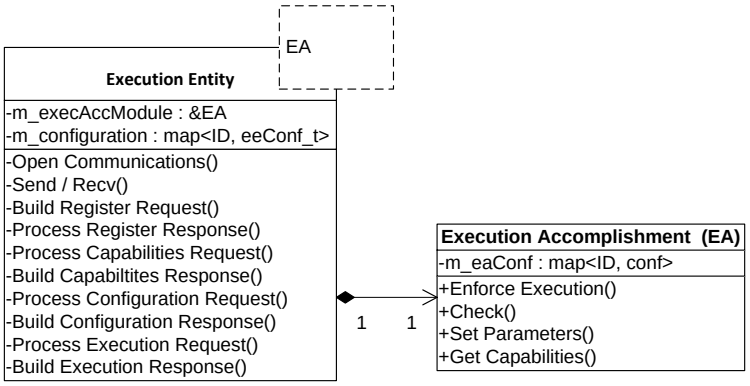


Figure 3.2: Structure of the generic EE object provided by the OConS library

ter (which is, in fact, yet another class, Data Source (DS) and Execution Accomplishment (EA)) tailored for particular purposes.

From a developer point of view, the template class abstracts the OConS related aspects, since he/she only needs to implement the action to be carried out. Furthermore, this abstraction would also facilitate migration of legacy connectivity mechanism into the OConS framework, easing the update of current nodes to OConS-enabled ones.

Figures 3.1 and 3.2 show a set of methods related to communication purposes (i.e. creation and processing of messages) which make use of the generic protocol library previously described. The interface between the two classes allows being aware of the times when the messages shall be created, as well as their isolated creation, according to the particular configuration(s) of the entity.

Due to the wide range of entities which might shape an OConS mechanism instantiation, it becomes necessary to find out the configuration parameters which can be considered generic. Besides, the reusability of the entities might be as well appropriate. This implies that the same entity can belong to more than one mechanism and, therefore, it should allow multiple configurations. For these reasons, the configuration of these generic entities covers two main aspects: multi-configuration support and explicit generic configuration.

In this particular implementation, the generic entities allow multiple configurations, coming from either different DEs or the SOP, for network monitoring issues. Those configurations are stored (as depicted in both Figure 3.1 and 3.2), thus enabling the reusability of the entities by different mechanisms. It would be particularly relevant for an IE whose information could be gathered

by several DEs or for EEs, if different mechanisms are using the same protocols of networking functionalities (for instance, resource management tasks).

Furthermore, the configuration within an entity takes place in two different places. Upon receiving a *ConfigurationRequest*, the outer template class takes the basic configuration parameters and forwards (calling the *SetParameters* method) the message to the inner class, which takes mechanism-specific parameters. The particular configurations for the two generic entities support are sent as a specific TLVs sequence (*ieConf_t* and *eeConf_t*), as described below:

- *ieConf_t*

  Parameter *notPeriod*: it establishes the notification period after which the IE has to inform the applicant (either DE or SOP) about the current information metric. This parameter is taken by the outer class and when the period expires, the *FillInPacket* method is called.

  Parameter *threshold*: it fixes a threshold (as a relative variation of the corresponding metric). Since the metrics of different entities would be rather different (for instance, in terms of data types), this parameter is handled by the inner class.

  Parameter *checkPeriod*: this parameter establishes the time interval used to check whether the metric has exceeded the threshold. Once this timer expires the *RefreshData* method is called, to check whether a threshold level has been surpassed, in which case the *FillInPacket* method is called to notify the requester.

- *eeConf_t*

  Parameter *numAttempts*: it fixes the maximum number of execution attempts the EE must perform before sending a *ExecutionResponse* after receiving an *ExecutionResquest*. This parameter is used by the outer class, together with the *Check* and *EnforceExecution* methods so as to be aware whether an execution has been correctly completed.

In both cases (*ieConf_t*, *eeConf_t*), the message also includes a grouped TLV, so as to identify the list of particular pieces of information the subscribed entities would like to receive (for the IE), or to gather 'tailored' information about the status of a particular execution, for the EE case.

3.2 Use Case: Orchestration of Two Mechanisms

The aforementioned development has been tested by means of a demonstration which comprises two different mechanisms so as to offer an enhanced access connectivity service, complemented with a dynamic mobility management. The demonstration was shown during the MONAMI'12 conference [8]. In particular, the scenario, depicted in Figure 3.3, comprises two access routers (with their respective Wi-Fi access points) and a mobile node which accesses a video server through the aforementioned routers.

First and foremost, the enhanced access selection mechanism is able to collect information from different network elements and it performs the decision according to a multi-criteria function, which is implemented at the mobile node.

On the other hand, the dynamic mobility anchoring mechanism offers a dynamic allocation of mobility anchors in the access nodes. The main idea is carrying out direct IP routing of the traffic flows initiated at the current access router, while forwarding only those flows which were previously started (different access router) using IPv6 tunnels. A brief description of the two mechanisms is given below.

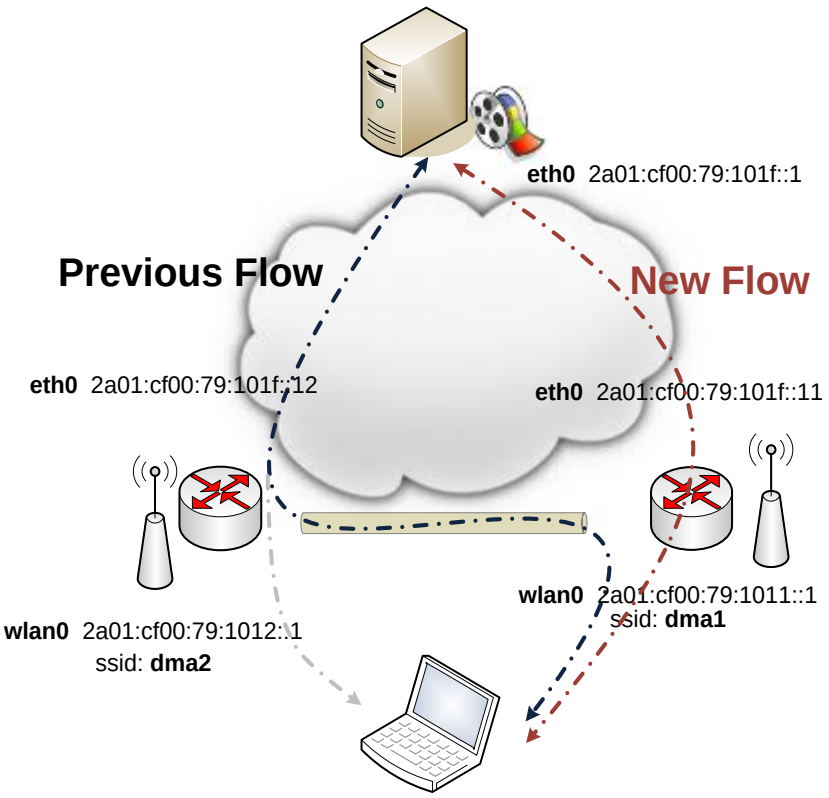


Figure 3.3: Orchestration scenario

3.2.1 Enhanced Access Selection

One of the goals of this implementation was to exploit the benefits the generic implementation brings about when deploying a new OConS mechanism.

From an OConS perspective, the mechanism consists of a DE which retrieves information from some IEs and makes use of an EE to connect to the selected access point. Concerning the orchestration tasks, both the OR and the SOP have been implemented, by using the different orchestration functionalities described in Section 3.1.2 while all the communication events between entities (or between them and the INC) make use of the generic OConS protocol library. Besides, the IEs and EE employ the generic template entities, which have been specialized, with the corresponding inner class implementations, in order to adopt the required behaviour.

According to the OConS framework description, all nodes in the demonstration scenario implement orchestration functionalities, implying that they include (at least) the INC, the OR and the SOP. The mechanism is embodied within the DE and EE located at the mobile node, and the IEs which provide contextual information, in order to enrich the decision procedure. The various IEs which were used are described below.

IE_{LQ} : this entity scans the Wi-Fi radio channels so as to inform its subscribers about the link quality (Signal to Noise Ratio (SNR) or Received Signal Strength Indicator (RSSI)) of the available access points. It is located at the mobile node.

IE_{UP} : in order to provide a tailored access, according to the type of user currently associated to the mobile device, this entity informs its subscribers about different user parameters, such as allowed, forbidden or favourite networks (according to the policies that the end-user can

establish). As the previous one, this information entity is also located at the user terminal.

IE_{TL} : this information module keeps track of the traffic load currently carried by the access routers; it is located at the network side and it is accessible by any mobile node connected to either of the two access routers of the scenario.

Based on the information collected from the IEs, the DE is able to make different decisions, according to the type of the current user. The basic decision algorithm has been implemented as a weighted linear function, which assigns different weights to the link quality (from IE_{LQ}) and traffic load (from IE_{TL}) metrics, according to type of user[1]. In this sense, the algorithm is able to give preference to the Quality of Experience (QoE) (link quality) perceived by the user (i.e. when she has some priority, business user) or to improve the load balancing, by fostering the traffic load parameter.

Besides the integration of the different generic components presented in the previous sections, a visualization tool has been developed so as to allow the tracking of the different procedures which take place during the decision process, including the different messages which are exchanged between the architectural components, their content, the included packets and their headers.

The following subsections depict some further details about the implementation, they discuss the goals which were achieved and they also provide an outlook to the work which is left for further research.

3.2.1.1 System Environment

The whole demonstration set-up is based on Internet Protocol Version 6 (IPv6) communications and its functionality has been implemented in $C/C++$ programming language. The mechanism has been distributed between the different network nodes (according to the particular location of the different entities). In particular, the mobile node has a *Linux/Ubuntu* (stable version 10.4) operating system, while the access routers are deployed in virtual machines, using $Fedora/Linux$. The network system environment and the devices which are deployed on the access routers will be afterwards described in 3.2.2.

It is worth highlighting the procedures that take place within a node to communicate entities and the INC. This has been carried through IPC communications, in particular by means of *Unix Domain Sockets*, since the inter-node communication uses regular UDP/IP sockets.

3.2.1.2 Realization Details

The enhanced access selection mechanisms is completely based on the templates of the generic elements which have been presented before. Hence, the realization details which are reported below are just focusing on the particular functionalities of the corresponding IEs, i.e. how they obtain the information which is then sent to the corresponding subscribers, the mechanism DE.

The mobile node is provided with an *Atheros* card (*ORiNOCO 11 a/b/g ComboCard*) by means of which connectivity takes place. This card is used by the IE_{LQ} to scan the radio medium and by the EE to establish the connections. Both entities interact with the card by means of *ioctl* calls.

Concerning the IE_{UP}, it uses a proprietary profile system based on tags which allows nesting user parameters. Finally, the IE_{TL} entity emulates the load at the access elements, offering a graphical interface to allow the user to change it[2].

[1] This is just meant to illustrate the possible combinations which can be used with the implemented elements
[2] This Graphical User Interface (GUI) was implemented in *Java* so, as it does not implement *Unix Domain Sockets*,

The DE implementation is slightly different; in this case, although it also uses generic procedures (described in 3.1.3), it requires specific modules for both the decision algorithm and the interaction with the other entities (e.g. scanning period or number of connection attempts).

3.2.1.3 Achievements

This implementation offers a proof-of-concept of the main OConS orchestration procedures and the corresponding communication processes and protocols. In this sense, it does not focus on a particular improvement of the connectivity. The main procedures whose operation has been assessed are: entities registration, capabilities discovery (both local or remote) and mechanism identification, validation and updating. On the other hand, the operation generic OConS protocol library has been also assessed.

Furthermore, this implementation has intensively used reusable and generic components for different actions, thus illustrating the openness and flexibility of the OConS framework, able to embed different connectivity mechanisms.

3.2.1.4 Future Reuse Capabilities

On the one hand, from the access selection mechanism point of view, it is worth mentioning that although it is rather orthogonal to other mechanisms, we have seen that the OConS framework can be used so as to smartly combine it with them (as it is with the Tunneling Management mechanism, which will be described afterwards).

On the other hand, the components which have been implemented are intrinsically reusable (from their design phase); in this sense, future implementations can take advantage of them, easing the process of developing new mechanisms, which will in addition share more homogeneous implementations.

3.2.2 Generic Execution with Tunnelling Management

The goal of this scenario was to dynamically activate the mobility anchors in the Access Routers (ARs), so that we were able to use direct IP routing of traffic flows/sessions initiated on mobile's current ARs, while supporting handovers and traffic forwarding/tunnelling between the ARs (current and previously used anchors ARs) for ongoing flows. Thus, within the OConS architecture, the focus was put on the Execution and Enforcement Entity in the mechanism realizing a dynamic tunnelling management approach.

The dynamic tunnelling activation is for interest especially for the flash crowd scenario; thus, the optimal data path is used each time a new flow is launched, the forwarding/tunnelling functions are activated only when needed to support flows' delivery in mobility situations, and the efficient connectivity with direct routing whenever possible.

3.2.2.1 System Environment

The code was developed in C under Linux/Fedora distribution (kernel 3.1.8) for the IP-related networking part. As for the radio part, we have used D-Link access points with Atheros chipsets and running the hoastap demon in version 0.7.3.

we have used the open source *JUDS* [10] package which provides this functionality

3.2.2.2 Realization Details

To implement the dynamic tunnelling mechanism, several software components were developed on the ARs and on the Mobile Node (MN). Accordingly, the software architecture is presented in Figure 3.4 for the AR, and in Figure 3.5 for the MN.

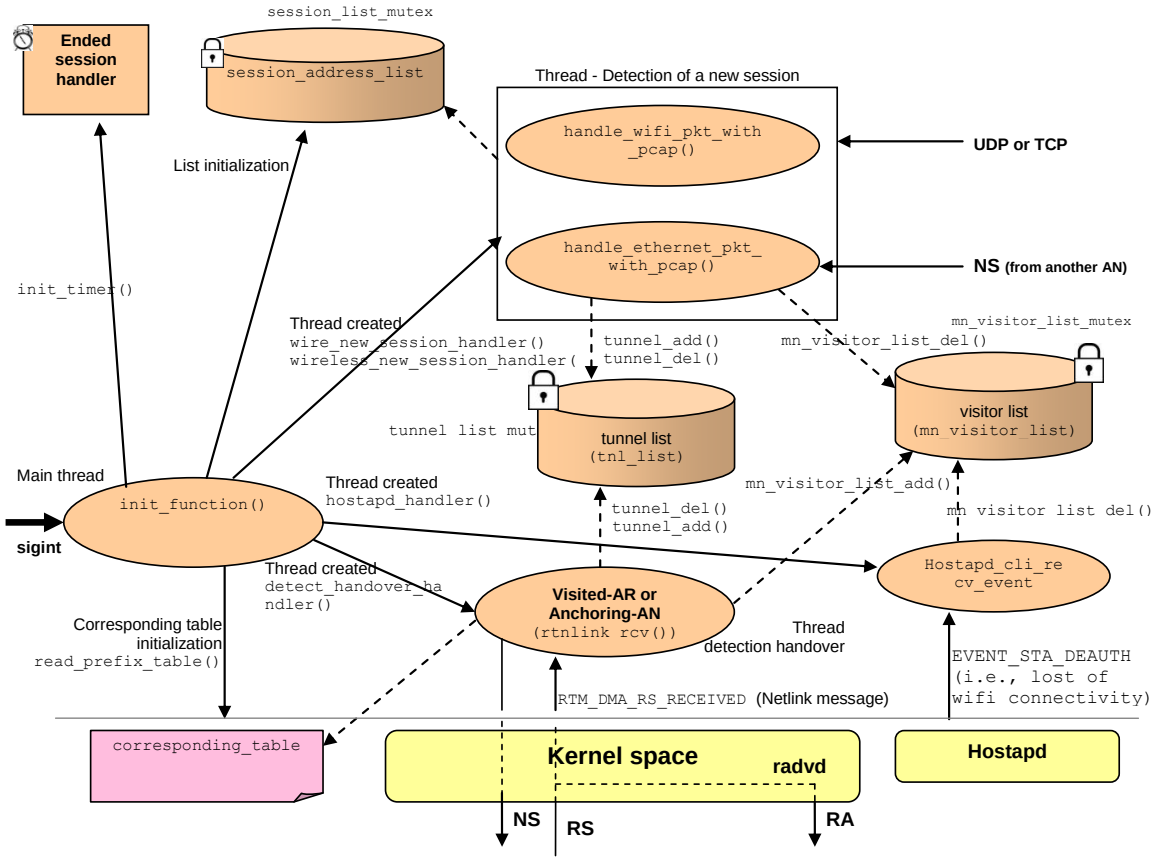


Figure 3.4: Access Router software architecture

Thus, the main components are:

- Detection of the new applicative sessions: On MN we use Linux Conntrack to track the connections and then in user space we have a function to wait for Netlink events from Conntrack; we also check who initialized the connection, the mobile node or the correspondent. On the AR, the detection of the new applicative session is done with the pcap library. Specifically, these events are handled in handleConntrackEvent() function from the dma_mn_new_session.c file on the MN, and, respectively in handle_wifi_pkt_with_pcap() from the new_session.c file on the AR.
- Detection and handling the handover (L2 and L3): The MN detects the radio handover at L2 (i.e., the new access point) by listening to control messages from Netlink framework, implemented in the new_handover_detected_by_newlink() function; then the MN informs the current AR by sending a slightly modified ICMPv6 Router Solicitation message and indicating the list of active applicative sessions (if any).
- Mounting the tunnels on the networking side and re-route traffic accordingly: On the current AR, once we receive the Router Solicitation we need to inform the previous ARs about the

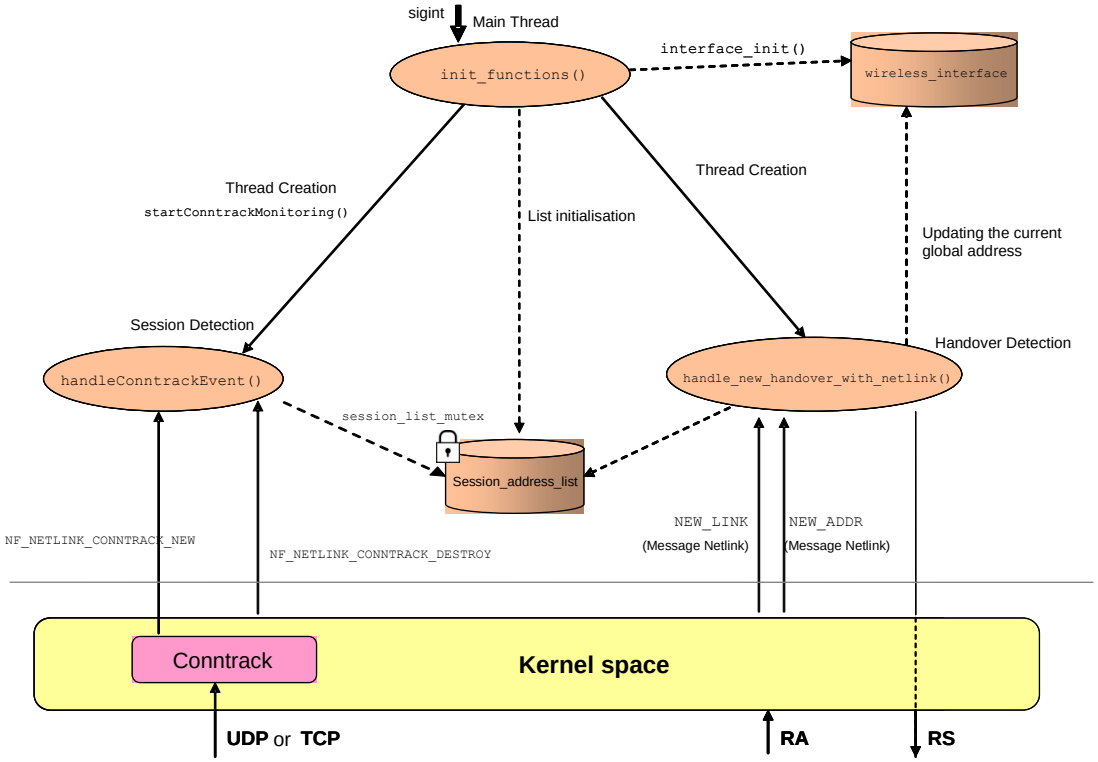


Figure 3.5: Mobile Node software architecture

handover and to create the tunnels towards these ARs. This is done with a modified ICMPv6 Neighbour Solicitation message, since we need to include and to signal all the current ongoing sessions established with the previous ARs. Then we build the new tunnel by adding a virtual network interface and by indicating the two end-points of the tunnel and by attaching the tunnel to a real Network Interface Card (NIC) (e.g., eth0). In addition, the routing is also update accordingly using the ip6tables and the ip -6 rule/route commands.

3.2.2.3 Open Interfaces and Interoperability with other Modules

The internal interfaces for this mechanism reuse existing protocols, such as ICMPv6-based Router Solicitation and Neighbour Solicitation messages.

As for the external interface, the mechanism is triggered by the loss of L2 Wi-Fi connection (i.e., the de-association event from the old access point and to associate with the new one).

Finally, note that this is done transparently for the applications' traffic, i.e., they continue to use the standard socket API without any modification.

We have integrated the dynamic tunnelling with enhanced access selection (i.e., mobility decision) mechanism using the OConS messages and the OConS orchestration functionalities.

3.2.2.4 Achievements

We have demonstrated our prototype in the following public events:

- IETF#83 public demonstration: The Dynamic Distributed Mobility demo was shown at the IETF meeting in Paris (during the DMM session), on 29th of March 2012.
- MONAMI 2012 public demonstration [8]: was demonstrated in combination with the enhanced access selection mechanism, Hamburg, 23-26 September, 2012.
- SAIL Final Even public demonstration: was demonstrated in combination with the enhanced access selection mechanism, Stockholm, 13th of February, 2012.

3.2.2.5 Future Reuse Capabilities

The mechanism with dynamic tunnelling for mobility management can be used separately from any other mechanism to achieve the IP-session continuity in mobile scenarios. The mechanism is currently automatically triggered by the loss of a link connection (e.g., lost of Wi-Fi connectivity), but it can easily be extended by adding any other event for triggering its execution (e.g. a command coming from a decision module or algorithm).

3.3 Discussion of Outcome and Results

It is worth recalling that the main goal of the prototyping activity which has been presented in this Section was to assess the feasiblity of the OConS framework. In order to tackle this objective, we selected two orthogonal, independent mechanisms and we exploited the OConS framework (the functionalities which were implemented) so as to smartly combine them (orchestrate). The outcome showed that, in principle, the proposed approach provides the flexibility degree which was aimed at. Although more work is required so as to extract more generic conclusions (i.e. testing with more mechanisms), the outcome of this activity provides a good basis.

The implementation which was carried out does not entail a remarkable complexity. There are some aspects which might be more difficult to deal with, like the dynamic reconfiguration of orchestrated services (in real-time) or the validity of the manifest to manage this configuration tasks. Another aspect which requires further work is the establishment of basic connectivity, which was not considered in the deployed demonstration.

Thanks to the reusability design which has been fostered, one of the major advantages of this prototyping activity is that it can be used so as to easily deploy OConS mechanisms and services. The developer would not need to worry about the way the supporting functionalities and the corresponding signalling protocol are called/used. This, of course, is also valid for different OConS services which might be integrated in the flash-crowd scenario.

Besides the already identified ones, there are as well other aspects which shall require more work in the future. The most relevant one might be the analysis of the scalability of the proposed framework; that would require adding more nodes to the scenario and see whether they have any influence on the perceived behaviour. Another key aspect (briefly introduced earlier) would be the integration of more mechanisms, so as to assess the flexibility and openness which should characterize the overall OConS framework.

4 OConS for CloNe: Elastic OConS Networking

The scenario "Elastic OConS Networking" deals with the interaction of OConS with CloNe showing how CloNe Datacentres (DCs) can use OConS to set up multiple virtual and physical networks, connecting them at the attachment points and using them as an 'elastic resource', with an abstract connectivity service between the attachment points and their 'single router' topology.

The scenario has been broken down to three major components that were realized in three stages: the datacentre interconnectivity, the flow-based domain connectivity control, and the elastic flow resource management.

The basic ideas behind this scenario are described in D.C.2 [5], Chapter 5 as well as in D.A.9 [2], Chapter 3. When it comes to the technical solutions applied from OConS, the reader is referred to D.C.4 [7], Chapter 3.1, and D.A.3 [11], Chapter 5.

Besides the OConS architectural issues, like OConS Application Programming Interface (API) and orchestration, this prototyping activity makes use in particular of the OConS mechanisms on Interconnectivity of Distributed Datacentres.

4.1 Interconnectivity of Distributed Datacentres

The Datacentre (DC) test-bed developed in SAIL demonstrates the possibility of defining and managing, using an OpenFlow (OF) based solution, the connection of different datacentre resources (such as hosts, switches, connectivity slices, etc.) over inter-domain environments. The objective of this demonstrator is to identify the gaps coming with a pure OF-based solution and how OConS solutions covers those gaps.

The datacentre test-bed is composed by two main elements:

- The Datacentre (DC) topology itself.
- The GUI that manages and monitors DC environments.

The topology implemented replicates the typical structure of a DC. It is basically composed by three zones (External zone, De-militarized Zone (DMZ) and protected Zone) which separate the functionality and services and restrict the accessibility to different machines.

4.1.1 System Environment

The topology used for the simulation of the interconnection of the Datacentres in the current test-bed is depicted in the following Figure 4.2 and Figure 4.3.

All the information needed to configure the topology of the DC test-bed is loaded in a XML file to build a dummy network (similar to the real remote environment), and then we collect variable information automatically from the real network.

As shown in Figure 4.4, the flows installed in the DC switches can be managed (viewed and edited) using the GUI.

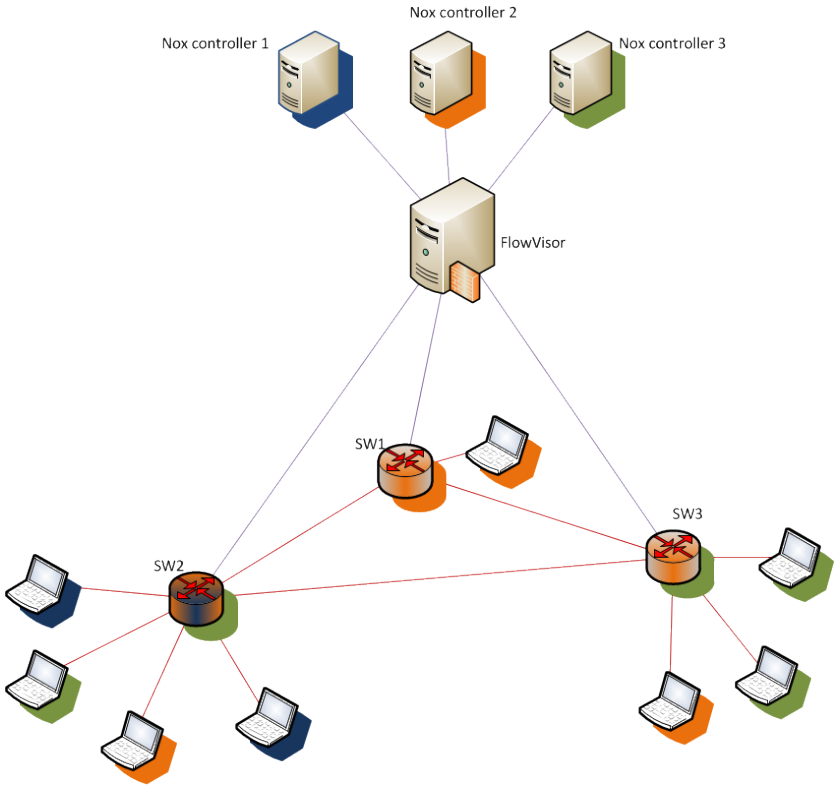


Figure 4.1: Slicing schema over the DC infrastructure

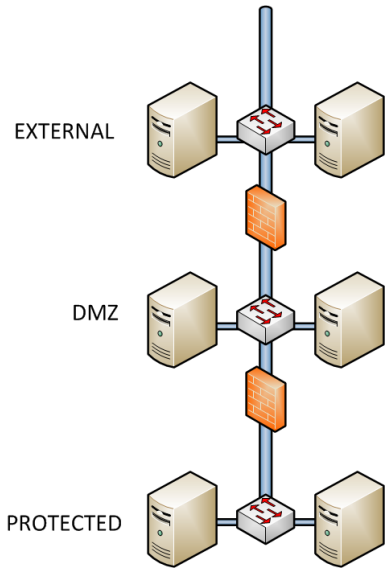


Figure 4.2: OpenFlow Datacentre client view

In addition to the OConS specific entities, the DC implementation includes the Distributed Cloud Plane - Link Negotiation Protocol (DCP-LNP) and the Infrastructure Service Interface (ISI)[1] interfaces specified and developed in the CloNe work package. In the current implementation, the

[1]The Infrastructure Service Interface is implemented using the Python Open Cloud Networking Interface (pyOCNI) library

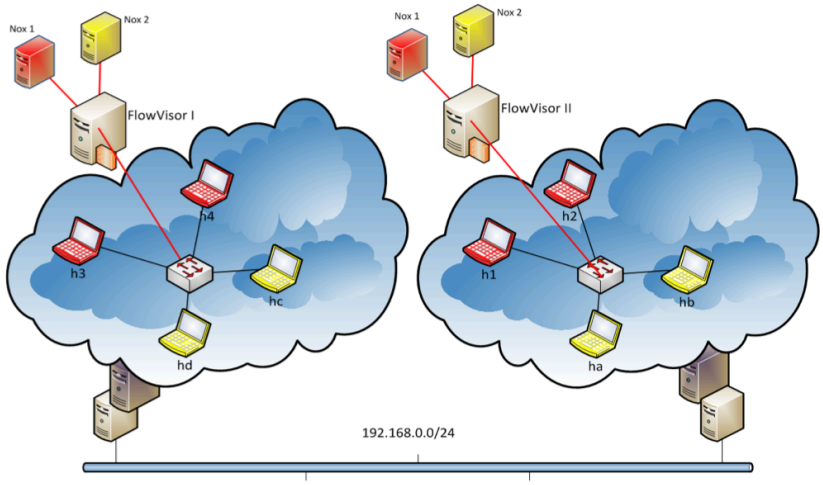


Figure 4.3: OpenFlow Datacentre interconnect

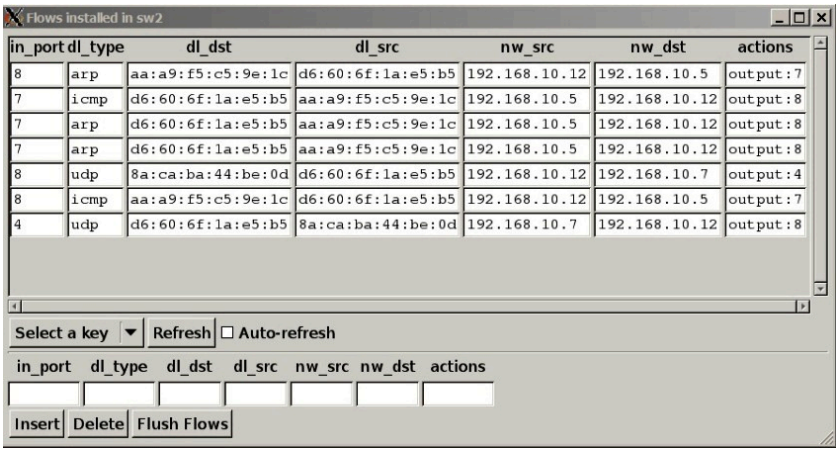


Figure 4.4: Access to the OF rules installed in an OF switch in the demonstrator

DC interacts with the OpenFlow v1.0 compatible switches.

The integration between the CloNe and the OConS world in the ISI is further described in D.A.3 [11]. Listing 4.1 shows a message to establish an OConS path using OF between two endpoints of the network test-bed controlled by the OpenFlow GUI.

Listing 4.1: ISI message to establish an OF path on the OConS infrastructure

```
{
    "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"]
        }
    }
```

In this example, the ISI interface in the DC demonstrator establishes a connection between end points 192.168.10.4 and 192.168.10.8 (addresses assigned to the PE interfaces), available in the mornings (from 8am to 12:30pm) and in the afternoon (from 2pm to 6pm), with a bandwidth of 100Mb/s, a maximum latency of 100ms and 0.01% loss. Profiting from the extra capabilities provided by OpenFlow, the connection is limited to TCP and ICMP flows. Additionally, the specification hints the use of an OConS-enhanced OF link as defined in the `OConSOpenFlowLink` mixin specification. The ISI definition allows the user to specify the nature of the connectivity between the end-points in a domain.

4.1.2 Functionality

The implementation of the OConS-enhanced OF Datacentre provides the full implementation of DC clients with three levels of isolation. Additionally, the OF switches can connect to other network infrastructure and further instances of the OF-DC. The networking part of the demonstrator can switch additional traffic that is not destined to the DC Virtual Machines (VMs).

This DC implementation is controlled by a specific GUI, that is deployed in a virtual machine. This virtual machine can be executed in any x86 compatible computer that supports VirtualBox [12]. This virtual machine includes a Mininet [13] network emulation environment. The OF Datacentre has been replicated on this network emulation environment. The DC control GUI is

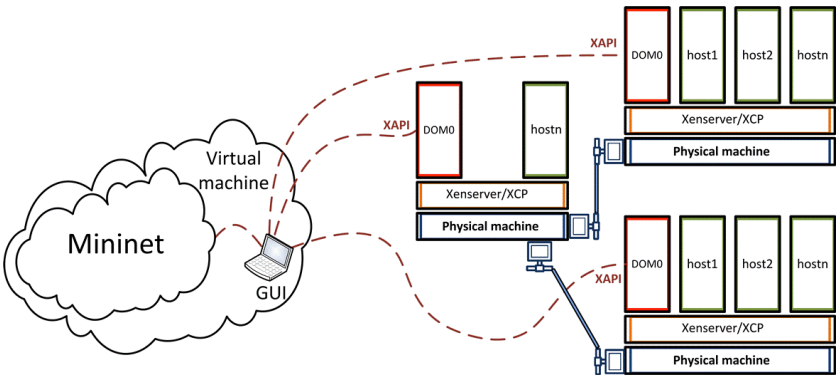


Figure 4.5: The full OpenFlow-in-a-box environment

able to control both the real and the emulated environments as shown in Figure 4.5.

4.1.3 Open Interfaces

The prototype implements an OpenFlow controller application that implements the REST-ful interface for the ISI and the DCP-LNP. This application is written in Python and executes on a NOX controller, which implements the OF protocol v1.0 towards the Open Virtual Switches (OVSes) both in the Mininet and in the test-bed. The test-bed can be accessed locally by connecting to spare FastEthernet cards.

4.1.4 Realization Details

Following technologies were used to implement the DC prototype:

- **OpenVSwitch (1.0.99)** is used as OpenFlow enabled virtual switch.
- **OpenFlow (1.0)** is used as networking protocol and provides separation between both data and control path
- **Xen Cloud platform (1.5 Beta)** provides an open source out-of-the box virtualization and cloud computing
- **Mininet (1.0.0)** provides scalable software-defined OpenFlow networks on a single PC.
- **NOX (0.9.1 Destiny)** is used as OpenFlow controller.
- **Floodlight (0.8)** is used as OpenFlow controller.
- **FlowVisor (0.8.2)** is a special purpose OpenFlow controller that acts as a transparent proxy between OpenFlow switches and multiple OpenFlow controllers.
- **Python (2.7.1)** is a programming language that lets you work more quickly and integrate your systems more effectively.
- **PyGtk (2.4)** easily create programs with a graphical user interface using the Python programming language.
- **Fabric (0.9.1)** is a Python library and command-line tool for streamlining the use of SSH for application deployment or systems administration tasks.
- **Json (2.0.9)** is a lightweight data-interchange format and it is easy for humans to read and write based on a subset of the JavaScript Programming Language

- **XML (1.0)** Extensible Markup Language that defines a set of rules for encoding documents in a format that is both human-readable and machine-readable.

4.1.5 Use Cases

The main demonstration use case for the OF-in-a-box environment is the datacentre interconnection use case defined in D.C.1 [14]. In the scope of SAIL, this use case has been used to demonstrate how next generation DCs can be deployed in a distributed manner along the network.

OConS components allow an extra level of functionality that complements the functionality provided by OpenFlow. One example is the possibility of monitoring the occupation level of the OF switching table. This table can only hold a finite number of entries. The IE contains historical data regarding flow switching behaviour. Additionally, the IE also holds utilization information regarding the switching table. This information that can trigger the aggregation of the flow switching table for a significant number of flows. This aggregation/deaggregation process is implemented in the EE.

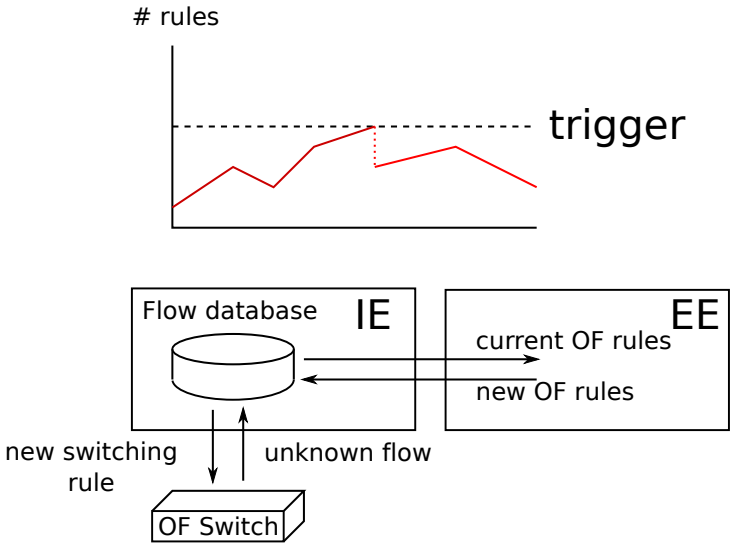


Figure 4.6: OpenFlow switching table control using OConS

This use case is shown in Figure 4.6. Enhancements to the OF protocol are implemented in the interface between the IE and the EE. These include the detection of flows that need additional rules to be treated by the OF switch optimally, sensing and management of the order in which the rules are applied in the OF switches governed by an OF controller, etc. The full implementation of this OConS application goes beyond the scope of SAIL.

4.1.6 Integration and Cooperation Aspects

The OpenFlow-in-a-box environment integrates CloNe functionality into the OConS Datacentre with OpenFlow prototype. This prototype has been used to evaluate the DC use case defined by OConS partners.

4.1.7 Reusability

The OpenFlow implementation of the Datacentre use-case is a corner-stone for the SAIL activities in OpenFlow both in OConS and CloNe. It is also used in partner internal demonstrations. Additionally, the fact that it is highly configurable and that the state of the OF switches is easily accessed makes it an ideal candidate for further OpenFlow seminars and training courses.

This prototype will also be used for proof-of-concept implementations for partner's activities in the Open Networking Foundation (ONF) and the Network Function Virtualization (NfV) study group at ETSI.

4.1.8 Achievements

The OpenFlow Datacentre has laid the foundation of the Software Defined Networking (SDN) research activities at some partners' organization for the next years. It is being demonstrated internally and at bilateral meetings with SAIL partners and other business partners in SAIL follow-up activities. It is the corner stone for several publications on SDN evolution (see e.g. [15]). The OpenFlow Datacentre has greatly influenced the development of Datacentre architectures at various partners and bears a great part of the responsibility for a shift towards OpenFlow and SDN in partners' views.

4.2 Flow-Based Domain Connectivity Control

Basic parts of this prototyping activity have been already described in D.C.3 [3], Section 2.1, but they are repeated here in parts (and updated) for clarity, readability and completeness.

4.2.1 Functionality

The prototype realizes experimental concepts for flow-based connectivity control per (network or technology) domain to be applied in multi-technology networks across layer3 routing, layer2 switching and in future even optical switching, especially with regard to control, management and algorithmic flexibility. It aims at providing a test-bed allowing experiments with new variants of path and resource allocation algorithms and their interfaces in a realistic network environment, demonstrating technical feasibility and gaining first performance measurements on processing delays, complexity, and capacities.

The architecture under consideration introduces a unifying control element in each supported network domain called the Domain Control Unit (DCU) that separates the control functions from the data forwarding functions by concentrating most of the control plane mechanisms and intelligence in a single entity. The DCU combines concepts of OpenFlow-based controller, Path Computation Entity (PCE) and Traffic Engineering Database (TED) for topology/resource advertisement and discovery as well as path computation and path/flow realization across multiple domains.

For experimentation purposes we follow an OpenFlow-like approach to realize a flexible and extensible test installation of multiple virtualized networking domains based on Mininet [13]. This enables us to demonstrate intra- and inter-domain path computation and flow establishment in a mixed environment of cooperating (layer2) switches and (layer3) routers in virtual and physical networks, as it is common in the context of interconnectivity of large distributed datacentres and cloud centres.

The demonstration in its first phase can show control and management of a data paths between endpoints in distributed remote (cloud) datacentres with inter-domain connectivity via web-based service and management interfaces from any browser (hence providing a REST-ful DCU interface for service invocation and orchestration), which can be used by CloNe users or the directly or the prototype for Interconnectivity of Distributed Datacentres as described in the previous section 4.1.

4.2.2 System Environment

This prototype consists of the DCU controller and the network emulation based on virtual switches configured by Mininet.

The internal functional architecture of the DCU and the interaction between its components is illustrated in Figure 4.7. As the DCU is hosting the main OConS functionalities (orchestration, DEs for mechanisms), we focus on the design and realization of the DCU, and use the available standard OpenFlow Switch implementations as example of a Domain Control Client (DCC).

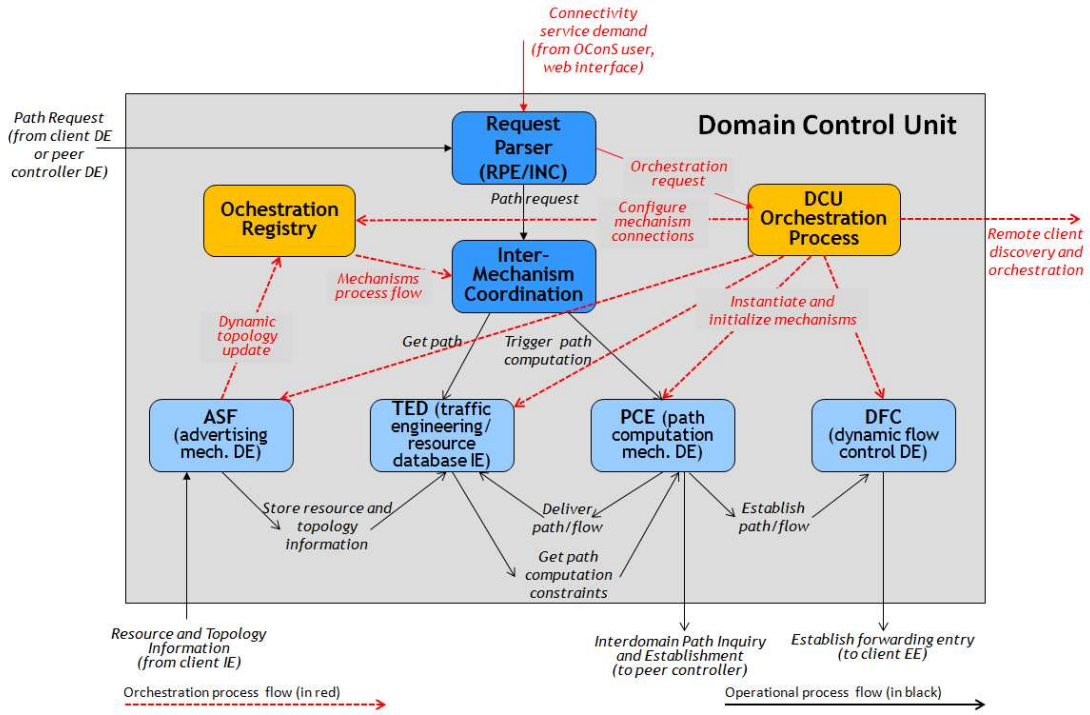


Figure 4.7: Domain control architecture - realization of internal DCU modules

The DCU is the domain-centric control server consisting of several internal components, namely Dynamic Flow Control (DFC), Advertising Supporting Function (ASF), Traffic Engineering Database (TED), Path Computation Entity (PCE), Request Processing Entity (RPE) and the external interface communication via OpenFlow protocols, REST-ful HTTP GUI and the Domain Control Protocol (DCUP) [16].

The RPE catches, interprets and coordinates incoming requests and converts them into the PCE protocol language if needed. It transforms the incoming requests into threads which keep the DCU and the TED stateless and allows for concurrent requests. Thus the RPE takes up the role of the OConS INC function. It identifies incoming messages and directs them to the appropriate DCU internal entities, e.g. to the PCE DE.

The functionality is able to process unknown flow requests as well as explicit path set-ups. In case of an explicit path set-up the DFC is directly triggered from the RPE to realize the path in the Network Elements (NEs). If an unknown flow request arrives and an explicit path calculation is needed to serve this request, the RPE triggers the PCE for path calculation.

The TED is the IE that persistently holds the current topology information, the current state of the network resources, traffic parameter, user/provider policies of the local network domain and pre-calculated (and continually updated) paths. In order to speed up the end-to-end path construction, the TED is configured to contain a set of pre-calculated inter-domain paths. In addition the the TED IE can store a set of calculated paths, which guarantees quick availability of these paths in case of failure.

The ASF retrieves topology and resource information from the IEs in the client NEs, which are supported from appropriate measurements in the NEs. It also recognizes initial set-up and dynamic changes in the domain topology and the availability of link and switching resources. Thereby the ASF supports the Orchestration function, e.g. during the start-up phase of the network it registers all DCCs of its domain. The ASF fills or modifies the appropriate entries in the TED. Beyond this, the topology and resource advertising mechanism can be triggered by event, periodically or on demand. In this way, the ASF assures that the TED is always kept up-to-date. However, for scalability reasons it is important to balance the actuality and the number of advertisings with an appropriate dynamic configuration of the advertising mechanism.

The PCE part as DE is responsible for the computation of intra-domain and inter-domain network paths. It is provided with current topology and resource information, with traffic parameters and management policies, and feeds the forwarding tables in the remote client NEs of the domain via the DFC.

In our realization, the DCU Orchestration Process (seen as an OConS SOP) is based on the Open Service Gateway Initiative (OSGi) software framework, which allows dynamic registration of software modules (i.e., mechanisms) as 'plug-ins' in the OR module on the fly, and connects and coordinates them via a service bus, which plays the role of the Inter-mechanism Coordination function.

The overall reference model and control architecture for this scenario was introduced in Figure 2.1 of D.C.3 [3]; accordingly, the placement of the OConS entities on the controller-side DCU and within the DCU's clients (DCC) is shown there. The overall system architecture follows a distributed, but domain-centric approach.

The OConS components and mechanisms used in this prototype realization can be described as follows:

- The DCU is responsible for the control of one OConS domain (cf. Chapter 3.2 from D.C.1-Addendum).
- Decision (DE) and information collection (IE) entities are realized in the DCUs placed in the connected OConS network domains, and if needed, within the (mobile) cloud datacentre.
- All three OConS entities (DE, IE, and EE) are used inside the clients DCCs of the respective domains.
- The DCU interacts closely with the connected DCCs to collect intra OConS domain information.
- The DCC comes in two flavours. The border node DCC (DCC-B) for inter domain connectivity and the domain internal DCC (DCC-I).
- The DCU also interacts with adjacent OConS domain DCUs to exchange inter-domain information, e.g. on processing resources available remotely or link load between such processing resources.

- Several attributes (Chapter 6.1.1 in D.C.1) such as resources currently offered or network quality of service (Quality of Service (QoS)), energy consumption and price (Chapter 6.1.3 in D.C.1) can be used by the resource allocation algorithm.
- When the appropriate information is collected by the DCCs and neighbour DCUs and a new processing task has to be carried out, the calculation and acquisition of appropriate processing resources and available network resources is performed by the resource allocation algorithm.
- When the network load changes, the DCU is able to react accordingly thereby offloading processing in overload or flash scenarios and providing a better data-path efficiency, service, QoS or other target the operator has specified. This ensures an uninterrupted service for the users and also an optimal usage of the operator's network resources. This is in particular addressed in the extended realization described in Section 4.3.

4.2.3 Realization Details

The DCU controller is realized in a Java/OSGi environment which is embedded in an Eclipse (Helios) Integrated Development Environment (IDE) [17]. This IDE runs on Linux as well as on Windows XP. In its functionality, it is based on and extending the OpenFlow Controller 'Beacon' [18].

The test network realization including the OpenFlow switches is realized by Mininet VMs [13] and virtual switches at distributed LINUX platforms (currently based on Ubuntu 10.04 LTS).

The network configuration used as a virtual lab test-bed is depicted in Figure 4.8. The configuration set-up was deployed on three physically connected (by 1G Ethernet) multi-core processor platforms running Linux Ubuntu 10.04 LTS.

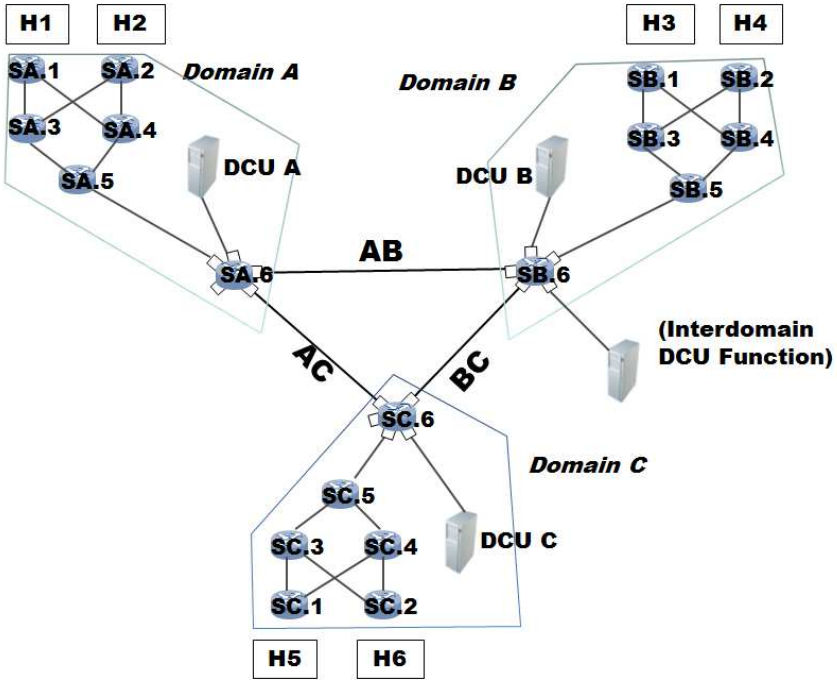


Figure 4.8: Network topology of the lab test-bed with 3 domains with labels A,B,C – 'H': denoting hosts, 'S': denoting switches

For further details, including external service/data, controller-client, and inter-domain controller interfaces, please see [16].

The following technologies were used for our implementation:

- *Open vSwitch* is a production quality, multilayer virtual switch;
- *KVM* (for Kernel-based Virtual Machine) is a full virtualization solution for Linux on x86 hardware;
- *Beacon* is an extensible Java-based OpenFlow controller based on an OSGi framework;
- *Mininet* creates a realistic virtual network, running real kernel, switch and application code, on a single machine (VM, cloud or native), in seconds, with a single command. It is easily possible to interact with the virtual network using the Mininet CLI (and API), customize it, share it with others, or deploy it on real hardware. Therefore Mininet is a great way to develop, share, and experiment with OpenFlow and Software-Defined Networking systems.
- *Eclipse*: a Java IDE supporting the development of any Java application, including Eclipse plug-ins (for OSGi and Spring);
- *OSGi*: The OSGi framework specification forms the basis of the Eclipse Runtime, fully based on the OSGi notion of bundle (equivalent to Eclipse plug-ins);
- *Spring* is a complete Java development framework. The Spring Dynamic Modules library (Spring DM) enables developers to build plugins with a consistent dependency injection across applications. Spring DM takes the service model of OSGi and simplifies it into Spring terms, making the dynamics of OSGi services a bit easier to work with.

In the following, we discuss some important details and aspects of the realization.

OConS Service Orchestration via OSGi

Beacon is a Java-based OpenFlow controller built with the OSGi framework. OSGi is a framework for Java services that allows different plug-ins to be installed, (re)started, and stopped at run-time. This feature is especially useful for an OpenFlow controller since it allows extension of the network management without the need to disconnect switches or even shut down the whole net. Beacon includes OpenFlowJ, a Java package that implements all specified OpenFlow messages and provides interfaces to create such messages to be sent to a switch.

Beacon comes packaged with multiple plug-ins that extend core capabilities of the controller, such as a plug-in for a learning switch that demonstrates how an OpenFlow switch sends unmatched packets to the controller which then decides the flow entry. In case of this learning switch a flow is created for the standard L2 and L3 destination of the packet.

Furthermore Beacon embeds Jetty, a Java web-server. Jetty is used to provide a web-interface for beacon, from which OSGi plug-ins can be monitored and controlled, and it gives an overview over the switches and their flows in the network.

The OSGi Framework implements a Service-Oriented Architecture (SOA), which has a high portability due to the Java technology. Functionalities of the components are provided as mechanisms (services) within the Beacon Controller or even to other systems. The services (plug ins) comprise arbitrary combination of IEs, DEs and EE. The service framework is the fundamental service platform including the Orchestration Register and the Orchestration Service Access Point (OSAP). The most essential functionality is the INC which provides a kind of Service-Bus connection which is used for the “inter-Plug-In” communication. This kind of Service-Bus is implemented as an Application Programming Interface (API) communication. Each service knows the Service definitions of the other provided services and is able to consume which is essential for an OR.

The service registry enables a bundle to publish objects to a shared registry (i.e., OR), advertised via a given set of Java interfaces. Published services also have service properties associated with

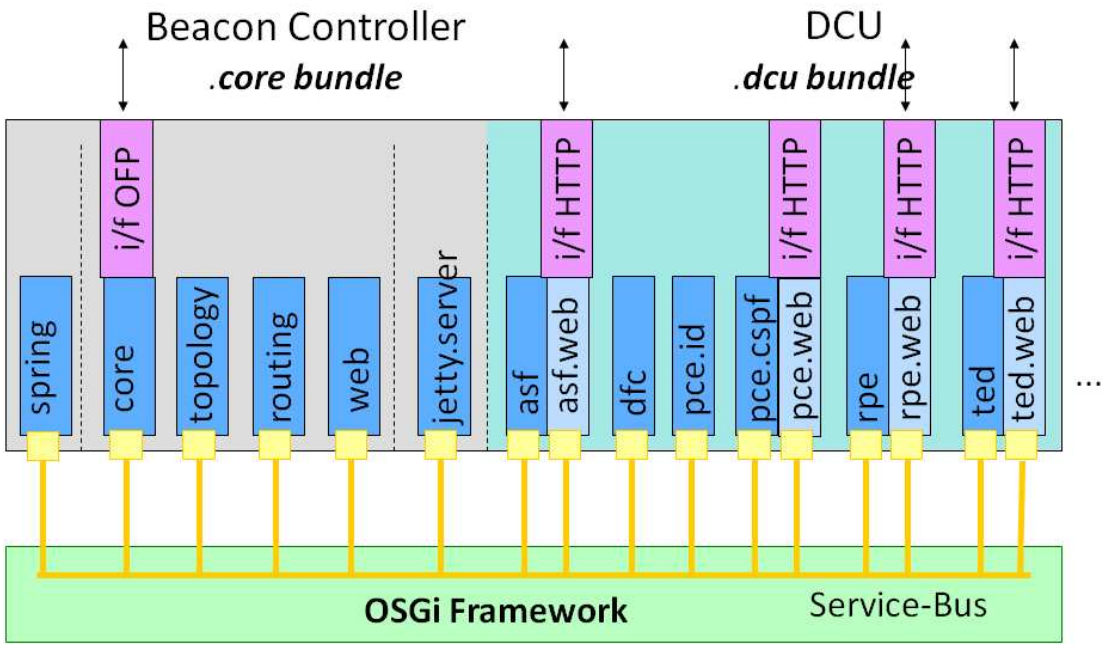


Figure 4.9: OSGi-based SW Architecture of the DCU

them in the OR.

While OSGi can support OConS service management in a more domain-centralized manner, it does not directly address distributed systems as SOA does. The framework handles mainly service components (i.e., bundles) and advertises the service interface (through the OR) of the components. The framework comprises some advanced features like dependency handling, multi-version support, and remote management to applications. Figure 4.9 shows the internal structure of the DCU plug-in mechanisms in the OSGi/Spring software environment, as grouped in the already available Beacon core bundle, and the OConS-specific DCU bundle. The support of the service orchestration process is a huge advantage, because the total Java application must not be restarted to update its components or to add new ones. The framework provides an Orchestration Register to enable services to register to bundles. Services are object instances of ordinary Java classes containing OConS mechanisms composed of IE, DE and EE components. The registration mechanism uses a service interface which is usually a Java-Type or a Java-Interface. To discover the service on the OR the interface or type is used as well. To support the service orchestration process dynamic, specific "Service Listeners" provide the option to react on registration or de-registration.

Any Java object can be registered as a service, but typically it implements a well-known interface. E.g. we show the registering of a very simple object as a service below:

```
Long      i     = new Long(20);
Hashtable props = new Hashtable();
props.put("description", "This an long value");
bundleContext.registerService(Long.class.getName(), i, props);
```

OSGi provides a very powerful local service and component model, however it does not directly address distributed systems as the Representational State Transfer (REST) approach does.

DCU API realization

The DCU process model is implemented using a REST-ful programming approach. REST is a lightweight alternative to Web Services and Remote Procedure Call (RPC). Much like Web Services, a REST service is:

- Platform-independent (i.e. need not care if the server is Unix, the client is a Mac, or anything else),
- Language-independent (C# can talk to Java, etc.),
- Standards-based (runs on top of HTTP)

A server response in REST is often an Extensible Markup Language (XML) file. However, other formats can also be used; unlike Simple Object Access Protocol (SOAP) services, REST is not bound to XML in any way. Possible formats include Comma-Separated Values (CSV) and JavaScript Object Notation (JSON).

A lightweight web framework *web.py* is chosen. Web.py is a public domain project following the Python philosophy of readability and ease of use. Web.py has built in support for REST-ful interfaces which allows to create responses to HTTP requests without the need to create any websites, the requested data can directly be send serialized as JSON or XML without the need to encapsulate it in HTML.

4.2.4 Open Interfaces

As depicted in Figure 4.7, the main interfaces between the DCU and the DCC are the following:

- External service interface: REST-ful control interface based on AJAX web technology, provides the GUI for the controller in any web browser (HTML, http, JSON coded).
- Inter-domain controller interface: uses the same data coding and functions as the service interface, just without GUI presentation. Can be invoked by functions like 'wget' and 'curl' with respective parameter sets.
- Controller-Client Interfaces: uses the OpenFlow controller to OpenFlow switch protocol V1.0 or V1.1.
- External data interface: data plane interface with IP-Ethernet stack.

4.2.5 Use Cases

The current prototype is able to demonstrate the following basic functionalities and show cases:

We show a virtual network test-bed with three network domains and their controllers, physically connected to each other via dedicated attachment points for control and data plane.

Furthermore, the OConS Orchestration mechanisms (including its Register and the Monitoring functions) are employed in an experimental way to support the controller in the Cloud datacentre interconnectivity use case.

Once the OConS node internal entities are discovered and the respective mechanisms are initialized, the Orchestration processes of other OConS-enabled nodes must be triggered.

During the initialization phase, all available network resources and capabilities have to be discovered and identified and have to be made available to the DCU via the node internal Orchestration functions. The discovery of a single resource, e.g. a switch node and its links or a general purpose CPU can run in parallel. The DCU can then start the algorithms and mechanisms that control the

used resources and assign resources to incoming service requests accordingly. Then, the DCU continually monitors the resources for their usage, e.g. for link, storage and CPU usage; for example, a smart access control and intelligent load balancing mechanism can improve the system response and resource utilization. Such mechanisms incorporated in form of a OSGi plug-in can be started and stopped manually at run-time by the management GUI.

Accordingly, the network operational phase allows the following actions:

Triggered by data traffic (via the external data interface), or managed by a web interface to the domain controller, flows can be set up, monitored and released within the own network domain (intra-domain), realized by a virtual network test-bed (including the switching/routing clients).

The domain controller monitors availability of any switching client and connecting link in the network domain, also the performance (QoS) of active flows, and it can rearrange flows in case of degradation.

In the inter-domain flow set-up, we use a separate 'umbrella' controller that only has knowledge about the inter-domain connectivity via the attachment points form the external view, and provides the necessary inter-domain controller communication (on domain membership of certain nodes, and possible sub-flows between the domain edges).

The controller web interface shows network states in terms of active flow tables, available network resource tables and visualizes actual flows on the network topology, for its own domain. The 'umbrella' controller shows the same features on an abstract inter-domain topology, without details of the involved sub-domains.

By orchestrating the plug-ins of the controllers, different routing algorithms can be activated on the fly such as 'shortest path', or 'constraint-based routing'.

It is also possible to manage (i.e. commission and de-commission single network and link resources) in order to test the behaviour of the mechanisms under consideration.

The focus in this prototype realization was to be able to test network-wide mechanisms for controlling the network topologies, the routes and paths and manage network-wide flows in a reasonably large multi-domain environment. As it was not possible to deploy such networks as physical networks with real switches and nodes, we used the virtual environment of Mininet for setting up larger configurations. The downside of this virtualized approach is that the performance is typically not good enough to operate the data-plane traffic for multiple network users.

To come to a realistic prototype using real processing resources in the cloud running on realistic hardware together with load dependent flow management on a production quality switch the *Elastic Flow Resource Management* demonstrator was conceived. This demonstrator is described in full detail in the next chapter.

4.3 Elastic Flow Resource Management

In continuation of the above introduced *Flow-Based Domain Connectivity Control* prototype, we developed a demo application casting the processing and switching of a video stream.

The difference to the previous demonstration scenario is that the focus has been shifted towards a set-up that was not only restricted to control procedures and protocols but was also able to handle data-plane flows with real-time processing in order to demonstrate the QoE of the OConS approach. To achieve this with available limited physical resources, the set-up has to be simplified in terms of involved number of switching and processing nodes. Also it was necessary to transform the realizations as described in Section 4.2 from the usage of the virtual Mininet environment to

the usage of single open virtual switches within a Linux kernel.

For the video processing an appropriate processing amount on certain processing units has to be reserved inside the Cloud Datacentre. Moreover the links to and inside the Cloud Datacentre service have to be established and monitored. All this is orchestrated and managed by an extended DCU. This DCU is the control entity that collects (IE) the needed information about current load of processing resources and link resources. Using a resource allocation algorithm the DCU DE decides on which processing resources the users are processed inside the Cloud Datacentre. Thereby the video flows can be switched over to other resources using other paths if needed. Connectivity changes are enforced (EE) by the DCU in the DCU clients.

In the demonstrator we cannot implement a full cloud data processing chain. Thus we choose the video processing analogue above to show that tight time constraints can be met and the needed processing units can be distributed remotely in the cloud and also connected in a flexible way.

4.3.1 System Environment

The modified system environment used is introduced in Figure 4.10. The set-up consists of a video server playing out the two video streams. The two different datacentres DC1 and DC2 are also referred to as Home Domain and Remote Domain and are connected via two steerable switches. Also the end users are connected in form of two VLC clients to show the video on the GUI.

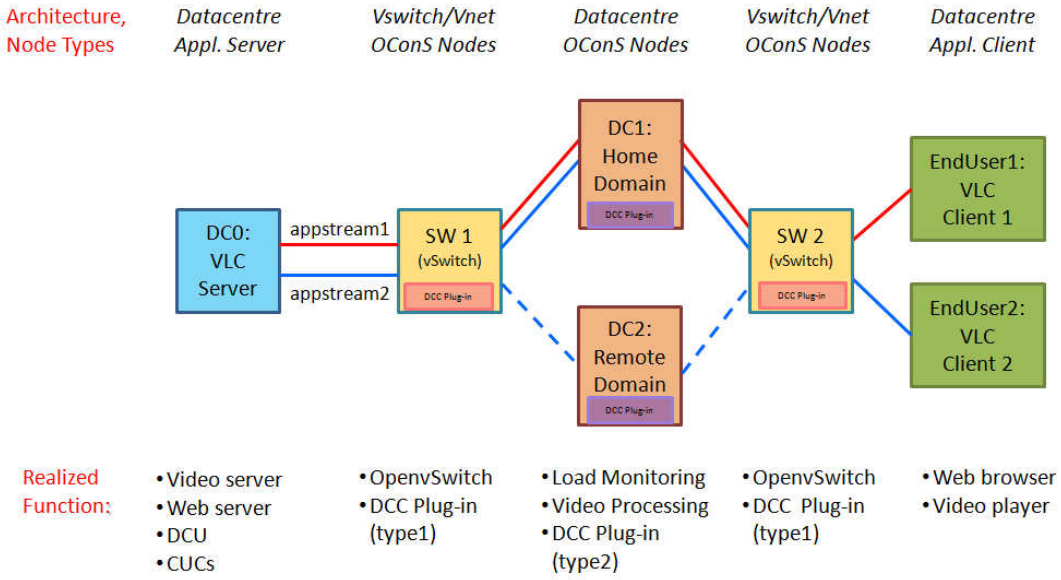


Figure 4.10: Logical Overview of the Prototype

The functionality and further realization details are described in the following sections.

4.3.2 Functionality

The prototype comprises load dependant flow and resource management to support service placement and real time video streaming. The chosen scenario consists of two video streams that are streamed from the same server to two different clients. On their way, the streams are processed by one of two machines that manipulate the video in a different and easily identifiable way. Initially both videos are routed through the same processing machine. As the load increases on this machine

(either naturally due to the video processing or due to load simulation) one of the videos should switch to the other machine, resulting in a different video outcome on the corresponding client. To connect the processing machines with the server and clients an OpenFlow switch is used on either side. This environment can be seen in Figure 4.10.

4.3.3 Realization Details

In addition to the tools and technologies described in Section 4.2, the implementation was extended by the following technologies:

- *Open vSwitch* is a production quality, multilayer virtual switch;
- *VLC Media Player* including the server and the client;
- *sFlow* is for monitoring physical and virtual server performance, to link network, server, and application performance and provide an end-end view of networked system performance;
- *Netem* (Network Emulator) e.g. for inserting delays into the network (delay emulation);
- *DoJo* toolkit for building high quality desktop and mobile web applications;
- *Raphael* is a small JavaScript library that enables work with vector graphics;
- *GStreamer* is a library for constructing graphs of media-handling components;
- *lookbusy* is a synthetic load generator;

4.3.4 Use Cases

The use case which is realized by means of this prototype set-up has been developed in the context of OConS support for CloNe, e.g D.A.9 [2]. The experiments conducted here were designed with special focus on the validation of the CloNe-OConS cooperation.

The end-to-end demonstration use case can be described as follows:

1. First video stream 1 is set up over the (default) home datacentre DC1.
2. Video stream 1 is fetched from the server and streamed over the home datacentre DC1 to client 1. Some distributed video processing will be performed at DC1 in order to emulate a certain cloud processing load state.
3. Second video stream 2 is initially set up over the home datacentre DC1 as well, using DC1 as the default route and processing path.
4. Video stream 2 is fetched from the server and streamed over the home datacentre to client 2.
5. DCC of home datacentre detects rising processing load across the default path using the Inter-DCC control protocol.
6. In the meantime, DCC of other remote datacentres (here the DC2) report free processing and network capacity using Inter-DCC control protocol.
7. Due to rising processing load on home datacentre DC1, the CUC at the DCU decides to switch over the end-to-end processing path from intermediate DC1 to DC2.
8. To execute this, the context of the video processing instance for the first video stream is transferred to remote datacentre DC2.
9. Existing video processing instance on remote datacentre is started with the new context of video stream 2 and becomes the video processing instance for the video stream 2.
10. Concurrently the video stream 2 is remotely switched by SW1 to run over remote datacentre DC2 towards client 2, invoked by the controlling DCU using the Inter-DCC control protocol

(as a variant of the OpenFlow control protocol).

4.3.5 Description of Prototype Components

The main components and their realization concepts were already introduced and explained in Section 4.2, there we only describe here the Extended Domain Control Unit used.

Extended Domain Control Unit (DCU)

The DCU has been extended by a bundle of mechanisms to explicitly monitor the processing load state on each intermediate node and modify the flows accordingly.

The DCU is the logical module that manages the monitoring and switching elements of the network using REST-ful interfaces. The DCU communicates with the DCCs on each switch. To communicate in a REST-ful manner with the load monitor on each processing machine the DCU contains Cloud Unit Controller (CUC) which adds the processing load management capabilities to the DCU. The CUC is implemented as a plug-in for the OpenFlow-based controller.

For each processing machine there is one service (WebServer) running on it and one CUC instance running in the OpenFlow Controller communicating to the remote service (WebServer).

The WebServer is responsible for monitoring the load of its machine and offering a REST-ful interface for retrieval of the resource information. This information is retrieved by the corresponding CUC so that the containing DCU has an overview over the processing on all involved machines. Thus the DCU has the complete overview on processing and link resources and can make the decision if one of the video streams should be switched to a different machine. If that is the case the DCU signals this to the CUC of the corresponding target processing machine and at the same time sends OpenFlow flow modification messages to the corresponding switches to reroute the stream away from its processing machine. To avoid conflict due to race conditions when the same switch is addressed each CUC has its own flow table number in a switch to insert and delete entries from it.

4.3.6 Prototype configuration

As virtualization infrastructure an efficient virtualization implementation was needed. As such the Kernel-based Virtual Machine (Kernel-based Virtual Machine (KVM)) was selected. Like the Open vSwitch it includes a kernel module which allows the virtualization scheduler to be run in kernel mode, greatly increasing its effectiveness. KVM itself however only handles the resource allocation for virtual machines, virtualization itself is done by Quick EMUlator (QEMU) an emulator. Both KVM and QEMU have the advantage of being entirely command line controllable which simplifies starting a virtual machine with complex parameters by enclosing it in a bash script.

The topology of the prototype set-up comprises two systems (Host1, Host2) emulating a data-centre running in a VM and an Open vSwitch that connects the VM to the outside network. The third system (Host0) runs the OpenFlow Controller as well as the video server. For video streaming and playing the VLC media player is used and for video processing a g-streamer instance for each stream on each datacentre. The complete topology can be seen in Figure 4.11 with DCC modules highlighted in turquoise and video processing components in blue (also see Figure 4.11).

The host systems are equipped with quad-core processors, each running two threads. Each VM is virtualized with two cores each running two threads. Additionally the first four threads of the host CPU are pinned to the VM, meaning that the VM will only use these threads on the hosts. This

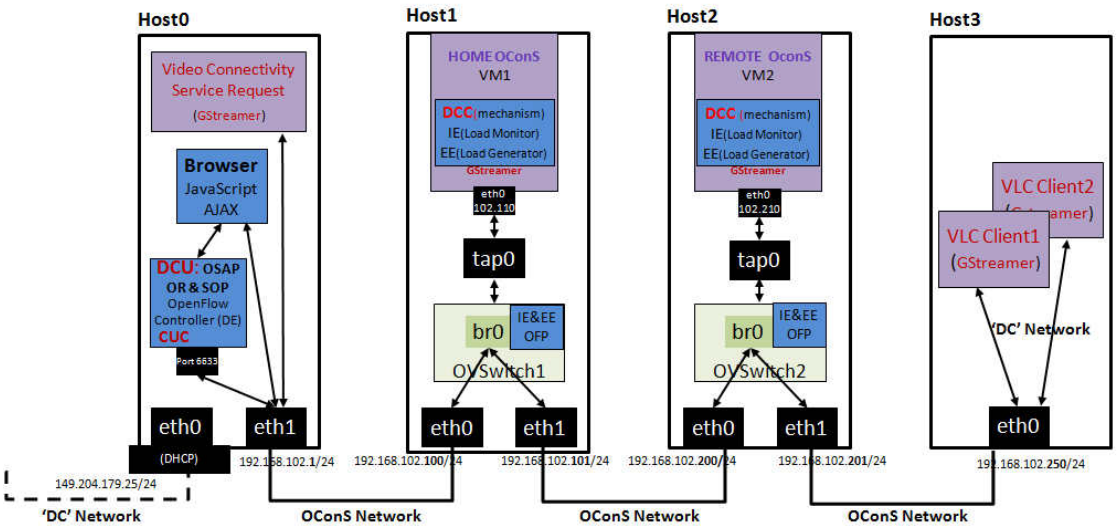


Figure 4.11: Elastic Networking Set-up

is done so the CPU load can also be seen from the host system on these four threads. Furthermore this ensures that even if the load in VM increases to very high levels there are still enough resources left on the host for the Open vSwitch. Each VM is also given 3 GB of RAM for the same reasons. Each VM is also started with one network interface which is mapped to a virtual tap interface on the host. A script to start the VM with these parameters can be seen in Figure 4.12.

```
taskset 0x0000000F kvm -smp cores=2,threads=2,sockets=1 -m 3072 \
-net nic,macaddr=52:54:00:00:20:00 -net tap,script=$PWD/ifup,downscript=$PWD/ifdown \
-drive file=/var/lib/libvirt/images/REMOTE-MSS-BBU.img
```

Figure 4.12: Script to start the Remote-datacentre virtual machine

Cloud Unit Controller Service

The CUC is the part of the DCU that is residing in the Beacon OpenFlow Controller. Each CUC is assigned to one datacentre. Its main responsibilities are tracking the number and properties of streams through the datacentre and balancing streams to neighbouring datacentres if the load on its own datacentre reaches a critical level. The CUC references its corresponding datacentre (virtual machine) as an OpenFlow device object. This allows easy access to all needed information (IE) to manipulate the flows in the corresponding switches. A CUC also maintains a list of other CUCs that manage neighbouring datacentres, which is used when the load on its own datacentre reaches a critical level. Each CUC is started in a separate thread when first activated. The CUC then continuously polls the load on its datacentre every 15 seconds using the REST-ful load monitor to see if the load as increased above the threshold. If this is the case, the CUC iterates through the list of neighbours polling each if they have free capacities. In this case the CUC prepares a set of *modify flow* messages to redirect the stream away from its own datacentre and messages the free CUC to take over the stream. After this, the first CUC waits for one minute before polling the load on the datacentre again. This is done due to the fact that the load from the datacentre is calculated by using a weighted average with a bias towards the present. By not using the current value during each poll, but an average of the last five minutes sudden peaks in the load can be

caught without messing up the load distribution. This however causes the weighted average to lag behind about 15-30 seconds, so the CUC waits with its next poll for one minute lest it pawns off another stream due to old information (see Figure 4.14).

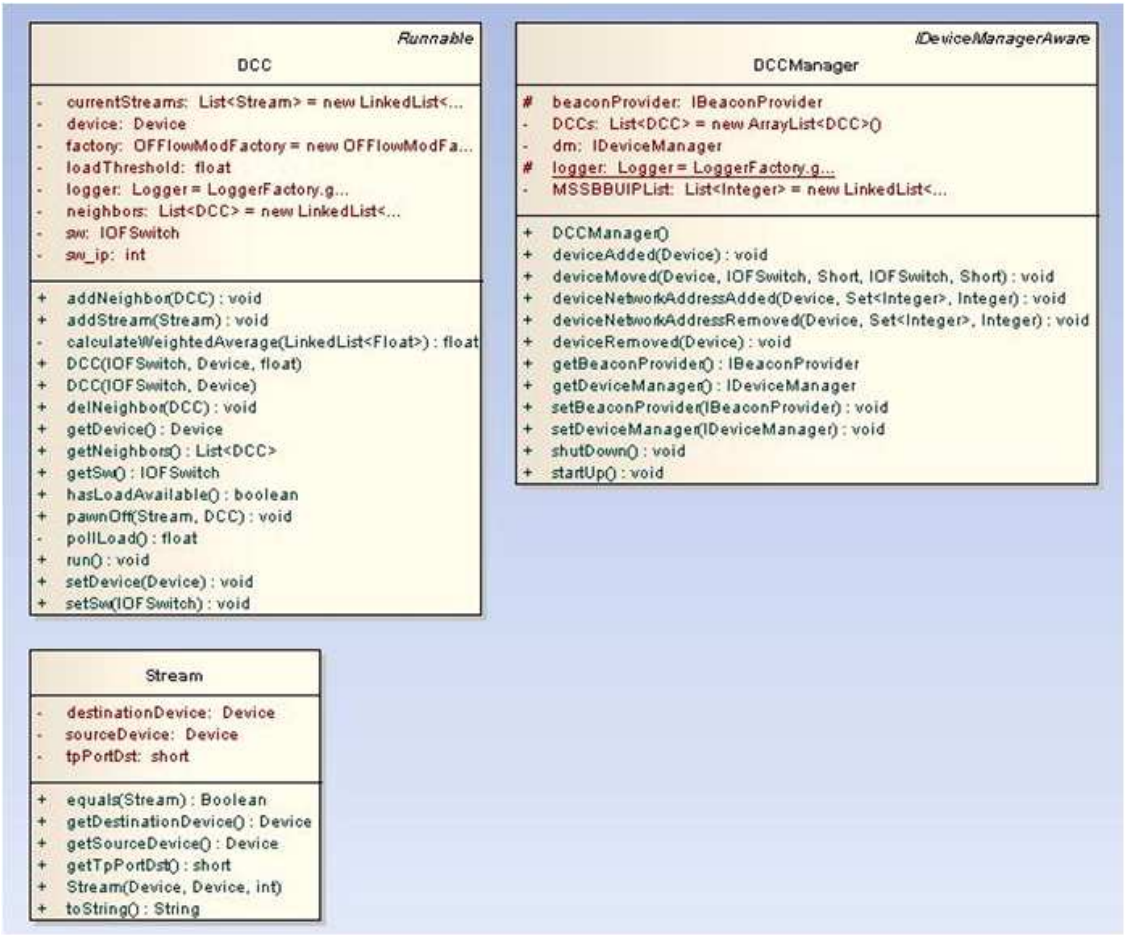


Figure 4.13: UML specification of DCC ClassObjects

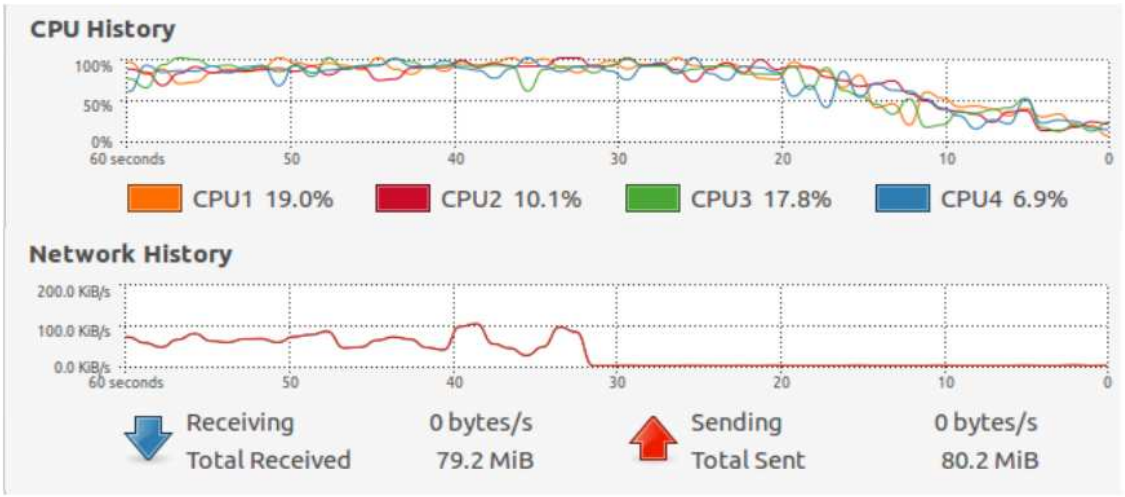


Figure 4.14: CPU load on the Home and Remote network controllers during a automatic run

Load Generator and Load Monitor Service

In order to impact and visualize the system behaviour, and enforce the use case above, a Load Generator and Load Monitor Service had to be implemented.

The Load Generator and Load Monitor offers a REST-ful interface to retrieve and manipulate the CPU load. An overview of the methods of the REST-ful API can be seen in Table 4.1.

URL	Method	Return Value
`/load/history`	GET	JSON File with the values of the latest 5 Minutes of CPU load, with 2 second interval
`/load/`	GET	last value of CPU load
`/load/<intvalue>`	PUT	sets the current CPU load to value of `<intvalue>`
`/load/<intvalue1>/<intvalue2>`	PUT	sets the CPU load to a curve with the parameters as minimum and maximum amplitude and a frequency of 5 minutes

Table 4.1: Load Generator and Load Monitor: REST-ful API

Load manipulation is handled using *lookbusy*. *Lookbusy* is a small Linux application that is able to set the CPU load of a machine to an arbitrary amount above the unmanipulated CPU load. Furthermore it is able to create a periodic curve for the CPU load between two values with an arbitrary frequency and manipulate only an arbitrary amount of CPUs.

For load monitoring the method reads Linux's */proc/stat* file every two seconds and extracts the amount of jiffies that passed since the start of the system and the amount of jiffies the CPU has been busy. These two values are then subtracted from their counterparts from the last iteration. The ratio between total and busy jiffies multiplied by 100 then yields the percentage of CPU load for the machine. This value is then appended to a queue which contains all values for the last 5 minutes.

The typical timing parameters can be changed by a management interface as shown in Table 4.1.

As an example, Figure 4.14 shows the system behaviour over time during the course of the use case with CPU loads and network throughput measurements on the home DC nodes, the curve at the remote DC being complementary (not shown here).

4.3.7 Reusability

Applying the OpenFlow protocol together with the state of the art Beacon controller facilitates interoperability and reusability among a wide variety of applications and tools. Moreover by using Open vSwitch, which is a production quality, multilayer open source virtual switch, the reusability is further improved. On the one hand Open vSwitch can run as a soft switch within the hypervisor. On the other hand it can act as the control stack for silicon based switches and has already been ported to various switching chipsets. Open vSwitch is open to extensions and more control not only using the OpenFlow protocol. Thus in the future a more sophisticated control mechanism will be possible. While the Open vSwitch fastpath runs inside the Linux kernel there are as well many userspace utilities available for it.

Also reusability and flexibility of our prototype components are further improved by means of REST-ful APIs. This is why also the load generator and the load monitor with their REST-ful interfaces to retrieve and control the CPU load are component candidates for high reuse. This also pertains to the graphical user interface module that controls and monitors the whole datacentre

and prototype environment in a web browser.

4.3.8 Achievements

The prototype validated the load dependent switching of different stream paths without any notable loss of QoE. Moreover the running prototype provides a proof of concept for the OConS functional architecture including the orchestration process and orchestration registry.

The described demonstration set-up has been presented live at the SAIL booth during the conference "Future Network and Mobile Summit (FuNeMS2012)", Berlin, July 2012. Also, the explaining paper [16] introducing mainly the case of "Flow-Based Domain Connectivity Control" has been presented there in a talk.

4.4 Discussion of Outcome and Results

The prototyping activities around the scenario "Elastic OConS Networking" have been broken down into three major components that were realized and studied in three stages: the datacentre interconnectivity, the flow-based domain connectivity control, and the elastic flow resource management.

The datacentre interconnectivity case demonstrated how OConS and CloNe actually can cooperate across a common Open Cloud Networking Interface (OCNI)-based interface, when OConS is used by CloNe as an underlying managed transport service beyond the basic IP capabilities. Thus, the SDN-like approach of OConS proved feasible and adequate in case of the limited scope of datacentre connectivity.

If it comes to the flexibility and extensibility of the OConS architecture, we dealt with a pragmatic alternative approach to mechanisms orchestration compared to the generic framework in Chapter 3. We developed a SDN-like domain control system based on mechanisms plug-ins for the (existing) service-oriented software environment OSGi, which allowed us to combine resource discovery and advertisement, route optimization based on PCE and TED functionalities with new OpenFlow mechanisms. The basic goal for these components were to complement the datacentre connectivity function as mentioned above. As the prototyping activities required design decisions early in the project, it was only partly possible to take into account all the architectural progress in later phases. This clearly points out some potential synergies, when merging the practical experience with the progress in understanding of the theoretical architectural framework, e.g. when further exploiting the mechanisms orchestration.

The extended experiment on the further tighter integration of OConS and CloNe for the optimized placement (and further on migration) of application processes based on the knowledge of available processing *and* network resources showed first results on the required monitoring capabilities and the exchange of data in both directions, in order to enable a cross-layer view of the optimization goals. This is in fact a tough theoretical problem, where we cannot give a final solution, but collected valuable know-how in retrieving and exchanging of required information. After the experiments, the OConS architectural approach seems to be flexible enough to cover such problems, too.

5 OConS for NetInf: Events with Large Crowds

This scenario shows how OConS can support NetInf through two mechanisms: Multi-path for Content Delivery and DTN Service for Retrieving Multimedia Content. Thus, for these two use-cases, we explain how the major components were prototyped, their functionalities in more detail, and we give a brief discussion on the results.

5.1 Multi-path Content Delivery for ICNs with OConS

Multi-path connectivity to networks in modern computing devices is becoming the norm in nowadays. These multiple paths can be used in many ways to benefit the users of these devices as well as the different service providers involved. The ICN architectures that are currently being defined, such as NetInf, have considered the use of multiple paths natively. Though these architectures provide the use of multiple paths simultaneously, no formal mechanisms have been defined to utilise them in the best possible manner. The framework and the mechanisms defined in the Open Connectivity Services (OConS) work package of SAIL project identifies a number of multi-path strategies that utilise the multiple paths to request and receive content with NetInf in the best possible manner. The prototype developed and demonstrated in this work shows the operations of the OConS framework and the related mechanisms.

5.1.1 System Environment

The prototype is based on the NEC NetInf Router Platform (NNRP), a NetInf prototype developed in the SAIL project. NNRP is a platform for developers to build and integrate modules to extend the functionality supported in NetInf. NNRP is a Linux based implementation. NNRP uses a module chaining mechanisms similar to iptables in Linux with a clearly defined interface for the different NetInf modules to interact with each other. The OConS enabled NNRP prototype for supporting multi-path content delivery consists of a number of NNRP modules. These extensions have the following features:

- Segment based content retrieval, where content is split into Named Data Object (NDO) and request pipelining is used to retrieve content;
- Use of IP networks as the underlying networking technology to request for and retrieve content;
- Network attachments of NetInf nodes are considered as individual paths;
- Use of the GET and GET-RESP messages of NetInf for content retrieval;
- Implementation of the Splitting strategy.

5.1.2 Functionality

The prototype uses UDP as the underlying transport network (Figure 5.1-(1)). It supports a number of different NetInf enabled applications including the NetInf enabled Video LAN Client

(VLC) video streaming tool. The NNRP modules for multi-path content delivery are as follows:

- vlc input - handles the requests for content from NetInf enabled VLC applications;
- vlc output - handles the serving of content requests of NetInf enabled applications;
- strategy - handles the Orchestration and DE functionality of the OConS framework to select the multi-path strategy;
- ocons - handles the interactions with the IEs to obtain information;
- nrs - handles the name resolution functionality;
- cache - handles the content caching functionality;
- mpudpcl - handles the EE functionality of OConS where the selected strategy is implemented.

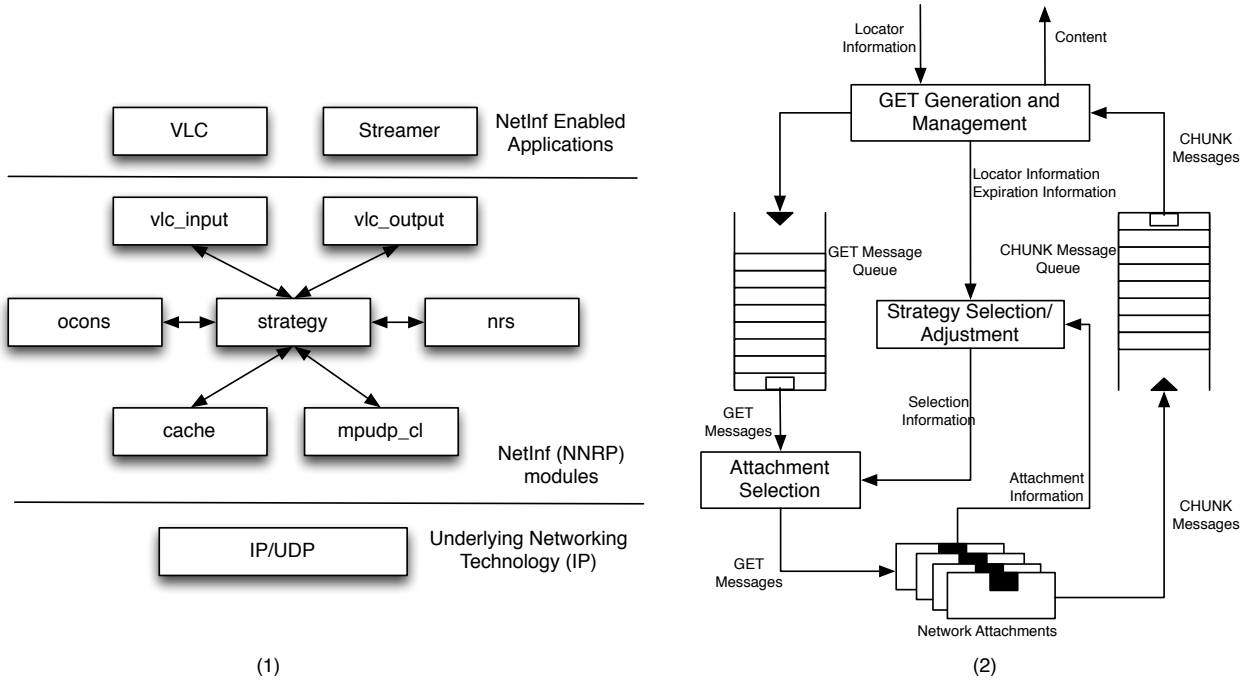


Figure 5.1: (1) OConS Multi-path Enabled NetInf Node Architecture (2) Splitting Strategy Algorithm

The Splitting strategy implemented in the NetInf nodes of the prototype utilises the multiple paths by distributing the NetInf GET requests for content into the multiple paths. This strategy uses the Additive Increase/Multiplicative Decrease (AIMD) approach to adjust the distribution of GET requests (Figure 5.1-(2)).

The OConS functionality in the prototype consist of the orchestration functionality and the use of the functional elements of the OConS framework. Most functionality related to OConS in the prototype is contained in the NetInf clients. The orchestration process bootstraps and sets up the client to perform the multi-path content retrieval. The OSAP requirements focus on using multi-path based on the delays experienced over each of the paths and the load information reported by the IEs in the network.

5.1.3 Configuration Parameters

The NNRP modules developed to handle the OConS multi-path mechanism have to be chained together in the NNRP configuration settings to pass the NetInf messages for processing. Following is a list of the main configuration information that can be varied to handle different environments:

- Maximum number of paths to use when using multiple paths to request for content;
- Communication parameters with the local instance of VLC (such the port).

In addition to the above mentioned parameters, further enhanced NNRP modules to handle different OConS based multi-path strategies can be easily integrated with, due to the defined inter-module messaging interfaces.

5.1.4 Open Interfaces

NetInf provides a number of message formats to communicate between different modules. 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:

- type - The type of the message;
- name - The content name;
- metadata - The list to hold additional meta data to accompany the message;
- length - The length of the data carried by this message;
- bits - The data that is is carried by this message.

GET Message

The GET message is used by the OConS Convergence Layer (CL) to receive and send the requests made for content. The GET message has the following format:

- type - Informs that this is a GET type message;
- name - Carries the string name of the content. This field may carry a NI based name in some cases, or a MPUDP based name in other cases;
- metadata - The metadata carries some information that is mostly used internally by the OConS CL functionality;
- length - This field is set to 0 in the case of GET message as no content data is present in a GET message;
- bits - No data due to the reason given in length field.

GET-RESP Message

The GET-RESP message is used to carry the content data by the OConS CL. The GET-RESP message has the following format:

- type - Informs that this is a GET-RESP type message;
- name - Carries the string name of the content. This field always carries a MPUDP based name;
- metadata - The metadata carries some information that is mostly used internally by the OConS CL functionality;
- length - This field is set to the length of the content field (bits) that follow;
- bits - This field carries the content data. The data is interpreted as binary data at all times.

5.1.5 Realization Details

The prototype has been developed and demonstrated in a Linux based environment. The following environment is required for the software to be operated:

- Compilation Environment: Requires GCC and the NNRP libraries;
- Runtime Environment: Requires a Linux 2.6 or above environment to operate.

5.1.6 Use Cases

The demonstration is based on the Event with Large Crowd scenario developed in the SAIL project. The scenario consists of a flash crowd that requires downloading content based on spontaneous decisions that are related to the interests of the participants of the flash crowd. The participants use NetInf based computing devices and some of these devices are deployed with the OConS multi-path extensions of NNRP. These devices are equipped with multi-path support and the operation of the OConS extensions demonstrate the use of multiple paths to deliver the required content.

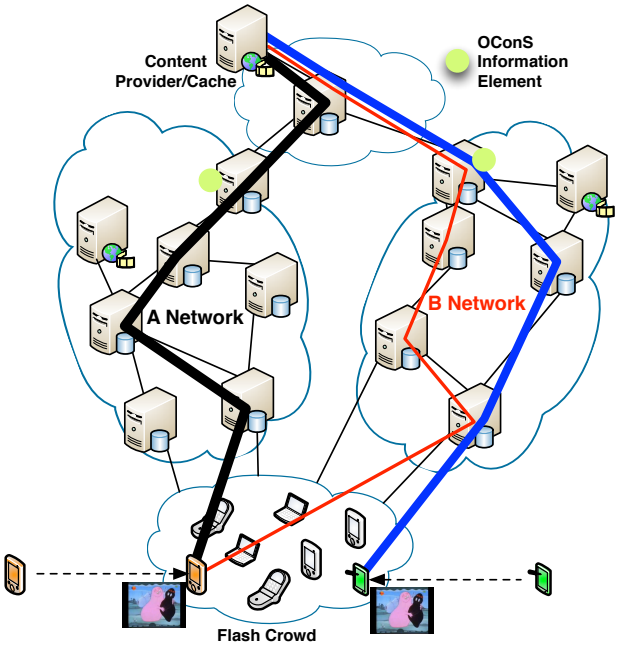


Figure 5.2: Demonstrated Use Case

Figure 5.2 shows the network topology used to demonstrate the use case in the public demonstrations we have shown. The focus of this demonstration was to show how OConS based multi-path content retrieval is performed when users join and leave the flash crowd.

5.1.7 Integration and Cooperation Aspects

The prototype is based on the NNRP implementation developed in WP-B. The multi-path extensions are implemented as a new NetInf convergence layer that can co-exist with the other CLs developed by the partners in WP-B.

5.1.8 Reusability

The NNRP modules that were developed conform to the interface defined in the NNRP platform. NNRP identifies a simple interface for the different modules to communicate with each other. Due to this adherence to the NNRP definition, the modules developed to support multi-path functionality can be used together with the other modules currently available or the future modules in the NNRP implementation. One example is the use of the caching modules. Currently, the simple caching module is used to perform NetInf NDO caching. But, a future more sophisticated caching mechanism that conforms to the NNRP interface can be used with the multi-path extensions.

5.1.9 Achievements

The prototype has been demonstrated in at public and internal venues. These demonstrations and the subsequent feedback have resulted in improvements to the multi-path extensions. Following is a list of these demonstrations.

- 2012 January - SAIL Plenary Sessions - Lisbon, Portugal
- 2012 September - MONAMI 2012 Conference - Hamburg, Germany
- 2013 February - SAIL Public Event - Stockholm, Sweden

5.1.10 Discussion of Outcome and Results

The prototype developed in this work was an attempt at evaluating the feasibility of the OConS multi-path mechanism for ICN, as specified in [5]. This prototype implemented one of the multi-path strategies identified but included most of the aspects related to the OConS framework. The focus of the prototype was mainly on providing multi-path content retrieval from a client perspective. These scope limitations helped us in developing a lab test-bed scaled prototype to evaluate the feasibility and to perform the demonstrations at public venues.

Considering experiences and the feedback received, this mechanism will be looked into in the following main areas:

- Extensions to include the use the other multi-path strategies identified;
- Extensions to include the operation of the mechanism also in network nodes (such as the location of OConS DEs and EEs in the network to make multi-path decisions and enforcements);
- Performance evaluation of the multi-path mechanism compared to the standard content retrieval mechanism adopted by NetInf.

5.2 OConS DTN Service for Retrieving Multimedia Content with NetInf

The HUman Routines optimize Routing (HURRy) protocol defines a probabilistic routing approach which infers and benefits from the social behaviour of nodes in disruptive networking environments. This OConS mechanism is intended for human carried devices (e.g. mobile phones interacting as DTN nodes), so that the dynamics and mobility of those DTN nodes can be translated into people's social behaviour that the routing protocol is able to use. HURRy is based on Probabilistic Routing Protocol based on Historical EncounTers (PRoPHET) [19] but it incorporates the contact duration to the information retrieved from historical encounters among neighbours, so that smarter

routing decisions can be made. The final purpose of integrating this mechanism with the NetInf implementation of Bundle Protocol Query (BPQ) is to show the cooperation between previous knowledge of social encounters among nodes and the possibility of intermediate caching of the demanded content.

This prototype is intended to show how several smart phones present in the Event with Large Crowd (EwLC) scenario could act as integrated NetInf-OConS nodes in such a way that whenever a user initiates a request (Query) for a multimedia content (i.e. photo or video file), the application interface sends a NetInf request ($GET\_req(content, location)$), and the local Service Orchestration Process is able to select a combination of BPQ+HURRy as the best service solution to serve that request.

5.2.1 System Environment

The environment consists of Android smart phones showing how the BPQ extension is used to retrieve multimedia files from a DTN node which has not access to a public server (using an intermediate cache). The system implemented for this prototype consists of several Android smart phones with a simple user application that allows to choose among four options:

- $DTNSend$: used to send either a video or a photo file to a specific DTN node;
- $DTNReceive$: used to invoke the retrieving of all packets addressed to local host from neighbour nodes;
- $Query$: used to send a GET_req message asking for a content file (without necessarily specifying a location or address);
- $Publish$: used to send a $PUBLISH$ message (broadcast) with a content stored in local cache.

The combination of both prototypes requires that the OConS DTN service incorporates the BPQ extension for DTN2, so that nodes are able to act as NetInf caches and to process BPQ requests and responses for the content shared among users. OConS DTN routing is implemented in Java code for Android, whereas NetInf uses a C++ implementation of BPQ, so some adaptation was needed to integrate both prototypes. If a certain smart phone present in the crowd was not able to use its Third Generation (3G) interface to retrieve a multimedia content from a specific public server, OConS would be able to orchestrate the instantiation of DTN connectivity service so that the node's request could be forwarded in a hop-by-hop basis until gaining access to the Internet. Moreover, if HURRy mechanism is available, DTN could rely on human-interaction based routing to better estimate which is the best neighbour node towards the destination address. And finally, if BPQ extension is also available, the requested multimedia file could be served from any intermediate node acting as a local cache, since any neighbour would first check if they have a local copy of the content before deciding to forward the request.

5.2.2 Configuration Parameters

There is no need for any configuration process to launch the user application on the smart phone, yet it is possible to perform some configuration on the inner function of the HURRy mechanism. Before the DTN2 protocol suite is compiled within the Java environment for Android, some parameters of HURRy can be tuned in order to vary the routing performance to be adapted to a certain scenario. During all connections with neighbours each DTN node infers and stores two important metrics: the contact frequency (inverse of inter-contact time) and the contact duration, which are used afterwards to estimate a weighted mean from the complete history of individual values. Assuming

both parameters are normalized to the same period in Equation 5.1, F denotes the inverse value of the averaged inter-contact time with a specific neighbour, and T stands for the averaged contact duration. HURRy was designed in a way that the relative weight assigned to each parameter in the probability estimation, can be configured by the system administrator. If $NodeA$ has its first contact with $NodeB$, their direct probability is initialized with a default value P_INIT (this value is set to 0.5 but can also be modified if desired). Otherwise, the DE derives the neighbour's quality by using the G (Goodness) formula:

$$G = \frac{F(T)^{1-\gamma}}{(1-FT)^{\gamma}}, \gamma\epsilon[0,1] \tag{5.1}$$

The goodness G of a neighbour is proportional to the frequency of contacts occurred (inversely proportional to the inter-contact time), and to the mean contact duration of past encounters. HURRy introduces a tuning factor γ in order to allow the administrator or the application to balance the priority among both parameters. It is easy to verify that when $\gamma = 1$ the frequency of contacts is being prioritized, whereas if γ takes values near 0 the goodness is prioritizing the contact duration. This will also influence the transitivity formula described by Equation 5.2 below. Equation 5.2 represents how $NodeA$ calculates transitivity probabilities to nodes at more than one-hop distance:

$$\left(\frac{1}{P\_(A,k)}\right)^{\frac{1}{\gamma}} = \left(\frac{1}{P\_(A,B)}\right)^{\frac{1}{\gamma}} + \left(\frac{1}{P\_(B,k)}\right)^{\frac{1}{\gamma}} \tag{5.2}$$

If we only considered contact durations (i.e. $\gamma \simeq 0$), transitivity would come from the minimum value of the comparison between $P\_(A,B)$ and $P\_(B,k)$. If we only considered frequency of contacts (i.e. $\gamma = 1$), transitivity would be given by the inverse combination of both probabilities. Since we introduced γ as a tuning factor, it also influences the combination law for transitivity, where Equation 5.2 provides a good intermediate law for the combination function. Once the value of γ is set, the administrator can compile the whole DTN2 protocol suite within the Java Development Kit (JDK) and produce the executable Android application with the desired configuration.

From the user's perspective, the input parameters required to execute the application's specific options will be either the name of a content file (to execute DTN_Send, $Query$ or $Publish$), or the destination address (only mandatory to execute DTN_Send).

5.2.3 Functionality

Connectivity in disrupted scenarios implies that nodes do not necessarily have permanent physical paths to certain destinations, but only to some of their closest neighbours instead. OConS has developed a mechanism that firstly registers valuable information parameters from the history of neighbour contacts, and secondly is able to combine them for an enhanced estimation of the contact probability in challenged networks formed by human-carried devices. The specification of HURRy is based on the outcomes of a thorough monitoring experiment [20], which highlighted the importance of distinguishing between short and long contacts and deriving mathematical relations in order to optimally prioritize the available routes to a destination in DTN environments. HURRy introduces a novel and more meaningful rating system to evaluate the quality of each contact in social deployments. Incorporating human behaviour to the routing decision has resulted in beneficial relations between common daily routines of people and the forwarding strategy implemented by their associated devices. That is the main functionality provided by the implementation of the HURRy protocol.

The implementation of the BPQ extension block to the Bundle Protocol allows extra functionality to the smart phones acting as DTN nodes: they can now perform intermediate caching and serve petitions locally without reaching the destination of the original request. Whenever a node of this prototype receives a certain content file (it does not matter if it was addressed to it or not), it will retrieve the archive and store it. If a certain content request arrives afterwards, it will be able to check if it possess a copy of that specific file and generate a response accordingly (without the need of forwarding that request to the final destination).

5.2.4 Realization Details

HURRy has been implemented as an evolution of the Bytewalla3 open source project [21], which is a DTN implementation for Android smart phones that incorporates PRoPHET as routing protocol. This prototype has evolved the available code so as to implement the HURRy protocol within the DTN stack available for Android. Apart from including the DTN service developed within OConS, the HURRywalla open source release (available at [22]) also incorporates the Publish/Query paradigm defined within NetInf. The BPQ extension block [23] for DTN has also been implemented over Android so that HURRywalla includes the possibility of dynamic content caching in Android phones. Although the realization of the described prototype has been implemented for Android, it could as well be exported to run on laptops with a Java runtime environment. The realization details are as follows:

- Operating System: Android 2.2 or above:
- Routing protocol: social routing as the HURRy protocol developed in Java for Android, taking the Bytewalla3 project release as a basis [21]:
- BPQ Extension Block [23]: BPQ functionality developed in Java for Android and integrated into the Bundle Protocol implementation of the DTN2 suite:
- Application service: multimedia exchange service implemented as a simple application for Android phones.

The complete record of modified and newly created classes in HURRywalla, with respect to the existing version of Bytewalla3, is listed in Table 5.1.

5.2.5 Use Cases

In order to present a showcase of the HURRywalla running operation, this section describes some functionalities trough the actual screen shot appearing in an OConS smart phone. Figure 5.3 represents the initial state of the Bytewalla application on Android, which has not changed in the HURRywalla implementation. The DTN service can be started by pushing the *Start* button, and the associated action is that the node initiates the neighbour discovery process via its wireless interface (HURRywalla makes use of WiFi over Android). On the right side of figure 5.3 the pop-up notice of the correct start is shown.

After initiating the DTN service in (at least) two smart phones, they start acting as OConS enabled nodes and exchange information regarding discovery and Routing Information Database (RIB) containing their probability estimations. Figure 5.4 shows the history of log messages captured by HURRywalla when a node sends/receives DTN Bundles. Once both nodes know each other, they launch the internal storage of inter-contact and contact-duration times, the estimation of all neighbours' probabilities and the exchange of their RIB with their peer.

After the information exchange has successfully taken place, the user can trigger any of the DTN

1. */servlib/routing/prophet*
   a) ProphetBundleRouter.java - MODIFIED
   b) ProphetNeighbor.java - MODIFIED
   c) ProphetBundle.java - MODIFIED
   d) BaseTLV.java - MODIFIED
   e) RIBDictionaryTLV.java - MODIFIED
   f) RIBInformationBaseTLV.java - MODIFIED
   g) OurProphetFunctions.java - CREATED
   h) RespActualProb.java - CREATED
2. */servlib/routing*
   a) BundleRouter.java - MODIFIED
   b) RouteEntry.java - MODIFIED
   c) RouteTable.java - MODIFIED
   d) TableBasedRouter.java - MODIFIED
3. */applib/types*
   a) DTNBundleSpec.java - MODIFIED
   b) DTNExtensionBlock.java - CREATED
   c) DTNBPQExtensionBlockData.java - CREATED
4. */apps*
   a) DTNApps.java - MODIFIED
   b) DTNSend.java - MODIFIED
   c) Publish.java - CREATED
   d) Query.java - CREATED
5. */servlib/bundling*
   a) BundleDaemon.java - MODIFIED
   b) BundlePayload.java - MODIFIED
   c) BundleProtocol.java - MODIFIED
   d) LinkBlockSet.java - MODIFIED
   e) PayloadBlockProcessor.java - MODIFIED
   f) BPQBlockProcessor.java - CREATED
   g) BPQCache.java - CREATED
   h) BPQCacheEntry.java - CREATED
   i) BPQFragment.java - CREATED
   j) BPQFragmentList.java - CREATED
   k) BPQResponse.java - CREATED
6. */servlib/convlayers*
   a) CLConnection.java - MODIFIED
   b) Connection.java - MODIFIED
   c) TCPConnection.java - MODIFIED
   d) TCPListener.java - MODIFIED
7. */servlib/discovery*
   a) Discovery.java - MODIFIED
   b) IPDiscovery.java - MODIFIED

Table 5.1: Classes in HURRywalla, with respect to the existing version of Bytewalla3

applications implemented in HURRywalla. In the left side of Figure 5.5 we can appreciate the four available options: *DTNSend*, *DTNReceive*, *Publish* and *Query*. As an illustrative example of one ad-hoc extension implemented in HURRywalla, the right side of Figure 5.5 shows the execution of a *PUBLISH* command as DTN application available within the user interface for Android phones. As described above, the *Publish* application provides a text box for the user to specify a destination address, but in reality (and because of the nature of a publication as defined in BPQ), it is spread to all neighbours detected. In this case, the user has chosen to share a photo file with his/her friends, but a video could have also been selected. Actually, HURRywalla provides the option of selecting a photo or video already present in the gallery, or to instantly take a new photo or record a new video.

Once the content has been successfully published, the receptor nodes of this shared file have a *dtn* folder created in their memory cards containing all received Bundles with non-empty payload. Figure 5.6 shows the structure of folders automatically created by HURRywalla, where two files are present: the bundle (text) and the payload (photo) stored. The user can open the payload file and see the photo recently published by his/her friend.

In the same way, a similar set of screen shots would show how a user could click the *Query* button

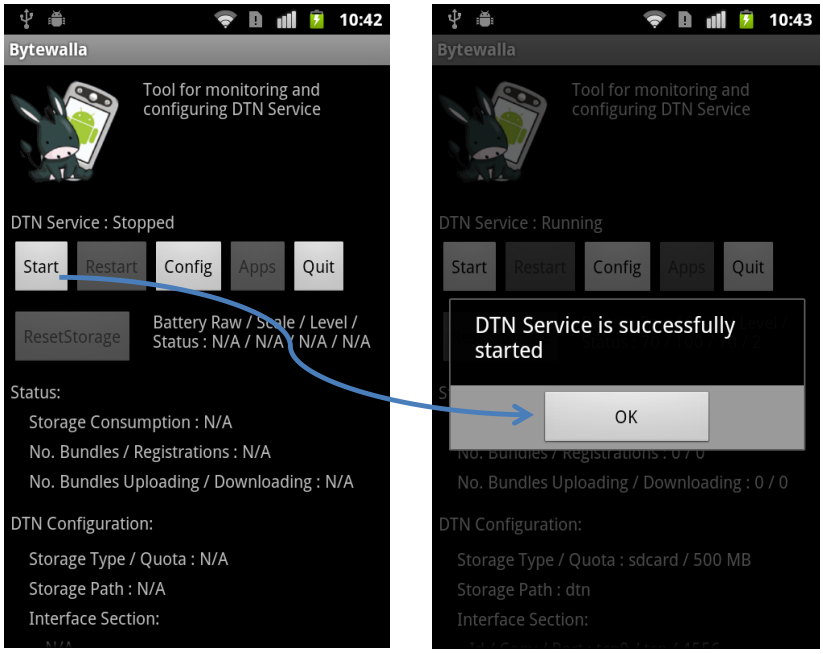


Figure 5.3: Launching the DTN service operation for Android: initial screen of Bytewalla

Figure 5.4: Notification of DTN routing information exchanged via HURRy protocol

and trigger a $GET\_req(content, location)$. If any of its neighbours would have the requested file stored in cache, the specified destination address of the GET command would not need to be reached, this neighbour node would generate a $GET\_resp(content)$ to the original requester. The OConS message sequence chart of this specific use case was specified in [5].

5.2.6 Integration and Cooperation Aspects

The history of required steps (milestones) that were planned in order to realize this prototyping activity within the time frame of the project can be summarised as follows:

- NetInf-OConS presentation and discussions during the first year, i.e., discussions and align-

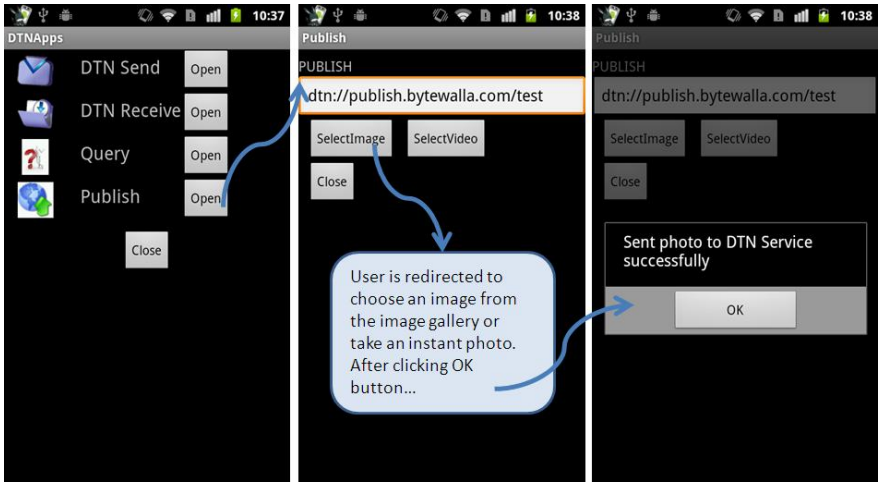


Figure 5.5: Operation of the DTN application for publishing content (photo or video file)

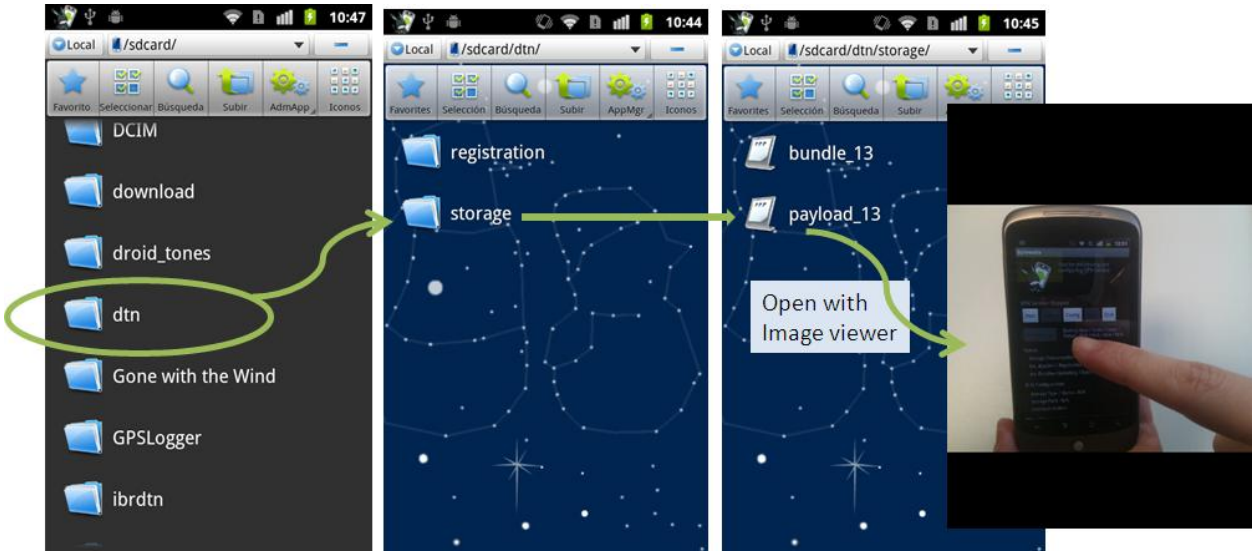


Figure 5.6: View of the content stored (photo file received)

ment of approaches, prototype goals and interface issues;

- Deliverable DA.9 (see [2]) documenting the interim state of NetInf-OConS cooperation plans;
- Work on the implementation of HURRy over Android smart phones during the second year, so as to serve the multimedia exchange application, based on BPQ extension. Discussions were maintained all along this period in order to combine social based routing with an adapted NetInf client for DTN environments (focus on EwLC scenario);
- NetInf-OConS face-to-face meeting and code sprint session during the SAIL General Project Meeting, Bristol, week of 17. Sept. 2012. We have finalized the descriptions of inter-NetInf-OConS prototype, interfaces and realization. We have aligned the use case realizations across WPs;
- Documenting the NetInf-OConS demonstration/prototyping realization in this present technical report.

5.2.7 Reusability

HURRywalla is the comprehensive implementation of the OConS DTN service described, including HURRy routing and BPQ extension over Android. HURRywalla has been released as an open source project evolved from the existing Bytewalla3 [21] and it is available from [22]. The whole code release can be reused and/or extended for specific user applications or with other routing protocols, bundle extensions, etc. The project is implemented in such a way that some parts of the DTN suite can be reused as individual modules as well.

5.2.8 Achievements

If DTN routing was to be implemented alone, the contents shared by people on the large crowd event could only be spread out on a source/destination basis. That is, a DTN node would require to know which node has the content and specifically ask for it to that source (either the content owner/generator or a node publishing its available content files). Combining the DTN routing with the NetInf caching on intermediate nodes, and using the BPQ request/response paradigm, a seamless communication is feasible. Furthermore, if BPQ extension is individually considered, the social behaviour of DTN nodes would not be exploited as routing information, which could result in a sub-optimal performance due to the demanding nature of a large crowd event. If some people participating in the EwLC belong to some social network and are connected with other friends, they would presumably have historical contact information about reciprocal encounters, and that is definitely useful for executing routing decisions in the depicted scenario.

5.2.9 Discussion of Outcome and Results

This evaluation work has been mainly focused on discovery protocols in DTN, as a test-bed to study the feasibility of the mechanism for optimised resource management and routing aspects, as well as adaptive self-management aspects, in topologies with resource constrained devices. We designed, developed, simulated and implemented a novel probabilistic approach for DTN routing and forwarding that exchanges information regarding nodes' social interaction and infers a mathematical relationship between these connectivity data and the estimation of how probable is that a new connection happens between two individual nodes. Transitivity is also incorporated in order to estimate multi-hop probabilities for new encounters. The whole solution has been derived as the HURRy protocol and it is meant to handle dynamics and mobility of the DTN network, using this information for the construction of the routing table. In order to validate the functionality of HURRy we performed a proof of concept evaluation using The One simulator (for opportunistic networks) (reported in [7]). After this simulation based test, we have also implemented HURRy on Android phones as part of the HuRRywalla project (see [22]) to verify responsiveness and transmission delays. Moreover, this prototyping effort has also served as an evaluation of the interface between the network layer (i.e. connectivity) and the application/service layer (e.g. NetInf). In our implementation for Android phones, we have also integrated the possibility of publishing and sharing multimedia content, by implementing the query/response paradigm over the DTN2 protocol suite. For this purpose, an existing DTN2 release has been modified and extended to use HURRy underneath, and the NetInf-BPQ based application layer.

The outcome of this experimentation work is very valuable, both on the dynamics management side, and on the content delivery validation side for DTN environments. The prototype verifies challenged topologies as an enabler for supporting a very demanding environment providing an alternative to traditional communication paradigms when these suffer from congestion or unavailability. In the field of DTN communications a lot of research is being performed, and this work is

part of the active field on innovative probabilistic routing approaches, whilst it also provides an open source implementation (released to the community) that allows further testing of publish/query requests and responses generation using the Android platform.

6 Conclusion and Outlook

The practical evaluation work in OConS started with the design and realization of a first experimental project phase, driven by early demonstration and experimentation activities of the partners accompanying the OConS technical framework definition, which resulted in intermediate demonstration show cases. These results were presented at the project-internal workshop in January 2012 (project month 18).

Based on these components and the gained experience, D.C.3 [3] described a further step of OConS prototype and demonstration activities which focused, in a second phase (project month 19-30), on the updated use case scenarios (from D.C.1-Addendum [4]), the work package internal cooperation based on the progress of the OConS architectural framework, and the cross-WP cooperation with CloNe and NetInf.

The final state of OConS prototyping activities as described in this document comprises:

- OConS reusable components for architecture and orchestration, such as the Generic OConS Protocol Library, which was assessed by means of the orchestration of two mechanisms; in particular, an enhanced access selection algorithm was combined with a distributed dynamic mobility procedure, featuring and improved tunnelling management;
- OConS support for CloNe with the datacentre interconnectivity, elastic networking features based on performance monitoring, path and flow management;
- OConS support for NetInf with the multi-path content delivery and the DTN service for retrieving multimedia content with NetInf.

One of the main objectives of the prototyping activities which were carried out during the SAIL project was to assess the feasibility of the proposed framework and mechanisms, and their contribution to the overall project-wide scenario of 'flash crowds'.

In the following, we highlight the most relevant aspects, which were undertaken in the OConS prototyping and demonstration activities, and their main results.

One demonstration [8] particularly focused on the OConS architecture, assessing the feasibility of the orchestration procedure. It included the barebones of the OConS architecture ([5]), specifically the signalling protocols, the INC, the SOP and the OR. In addition, it integrated two illustrative mechanisms which were used so as to show the potential, which OConS, in general, and orchestration, in particular, might bring about.

Thus, as a proof-of-concept, the *Enhanced Access Selection* mechanism was combined with the *Dynamic Distributed Mobility* mechanism to provide an OConS service; moreover, this dynamic tunnelling mechanism can be used by other decision entities from any other mechanism in order to achieve the IP-session continuity for a given flow. From a prototyping point of view, the development and the experimentation work on the mobility execution has allowed us to achieved a better understanding of the basic functional entities (i.e., EE), as well as to validate our approach and to demonstrate its feasibility.

The second scenario was grouped around the distributed datacentre connectivity as required to support the network service needs of CloNe. The combination of mechanism comprised of the consistent creation of virtual networks within and between distributed datacentres as the net-

work resources of CloNe Flash Network Slices (FNSs). The realization tested the feasibility of an OpenFlow-based approach in an SDN-like set-up. In this sense, it shows that the CloNe interface definitions for requesting the connecting network resources (encoded in OCNI format) are indeed matching the requirements for the OConS northbound API as defined in [5]. Both OConS and CloNe use a common REST-ful approach for the interface, which eases their communication at the control and management level, either by means of a GUI or a machine interface, emulating this mimic by HTML GET and PUT messages. Further discussions of the CloNe-OConS related can be found in [11].

The realization of an OConS domain controller (DCU) with the capability of orchestrating multiple network mechanisms is described in [16]. The principal functions of network and link state discovery, advertising and monitoring (IE functionality) were combined with intra and inter-domain path and flow computation (DE functionality), establishment and maintenance (EE functionality). The orchestration via the flexible OSGi SW framework in the OConS domain controller (typical SOP functionality) was able to register and integrate different path computation mechanisms (PCE alike) and store the network state in the TED.

For the 'elastic networking' functionality of OConS, it turned out that the 'delegation' principle of the resource management in the CloNe distributed cloud layer/manager had to be extended with a callback capability to the CloNe fault management, in order to close the feedback loop when the OConS-managed network is no longer able to autonomously provide the requested network and computing resources between the involved attachment point.

The other demonstration scenario dealt with the interaction of OConS and NetInf. The relationship between NetInf and OConS appears to be straightforward. OConS can be essentially conceived as a packet transport mechanism. However NetInf is, by virtue of the convergence layer concept, able to use specific packet transport mechanisms of different capabilities for the operation of NetInf over OConS. At the top end of a NetInf convergence layer, a service access point for the NetInf protocol has to be realized. Hence, and in order to ensure that OConS and NetInf complement each other, the corresponding convergence layer shall be developed.

OConS Multi-path Network of Information (OMPNetinf) mechanism extends the NetInf architecture to provide efficient multi-path retrieval of content through the use of the OConS Framework; it introduces a number of forwarding strategies (mechanisms) to retrieve content: splitting, replication and distribution. OMPNetinf utilizes the OConS Framework elements (such as orchestration to perform discovery and selection of OConS mechanisms and the coordination of operations) to select and use the appropriate multi-path strategy by means of IEs, collecting and providing information for the multi-path strategy, DEs selecting and configuring of the forwarding strategy and the EEs enforcing the decisions made by DEs.

In addition, Chapter 5 has also presented the implementation of a routing protocol for DTN which, thanks to the OConS functionalities is able to improve its operation. This can be of outer relevance for flash-crowd scenarios, since it is rather likely that nodes (due to intrinsic mobility) have highly intermittent connectivity patterns between them.

During the project, it turned out that the prototyping activities required early design decisions, and could only partly take into account the final state of the discussion in the OConS architectural framework. This clearly points out some potential synergies, when merging the practical experience with the progress in understanding of the theoretical framework, e.g when further exploiting the mechanisms orchestration. So the value of the the demonstration and prototyping results is more providing the playground for experiments rather than to come up with a complete single demonstrator integrating all aspects of the OConS framework. For the goal of re-organizing the transport level (as was the goal of OConS) that comprises so many diverse and heterogeneous technologies,

involving also a lot of specific physical hardware components, it would have meant both exceeding the limited resources of this project and diluting and concealing the successful practical progress if we had enforced the combination of all the achievements onto one platform.

All in all, we can state that the initial objectives of the prototyping activities have been fulfilled. As was reported in this document, the particular goals of each of the tasks which were carried out were successfully reached. Furthermore, all of them have been externally presented, in public demonstration events, as papers presented in conferences, or by means of publicly available code projects. The feasibility of the approach fostered by the OConS framework has been assessed, either as a means to combine connectivity mechanisms (see Chapter 3), or for specific use cases (functionality tailored for certain scenarios), see Chapter 4 for the applicability of OConS for CloNe and Chapter 5 for NetInf. With the outcome of these implementation tasks, the project has shown that the OConS concepts certainly provide a step forward in order to bring the SDN paradigm to real networking scenarios.

A Selected Demonstrations of OConS Features at Public Events

The achievements of the OConS prototyping and demonstration activities have been publicly presented at the following events:

- the conference "Future Network and Mobile Summit (FuNeMS2012)", Berlin, July 2012;
- the conference "4th International Conference on Mobile Networks and Management (MONAMI'13)", Hamburg, September 2012; and
- the workshop on "Future Media Distribution using Information Centric Networks", Stockholm, February 2013 (where an extended version of the MONAMI demonstration has been presented)

For information purposes, we include in this Annex the poster and brochure material that was used to introduce and explain the demonstrations.

‚Elastic Networking'
Open Connectivity Services (OConS) for Cloud Networking (CloNe)

SEVENTH FRAMEWORK PROGRAMME

GOAL
- Show principles of load-dependent resource allocation between application, cloud management (CloNe) and network (OConS)
- Control and management of a data paths between distributed remote (cloud) datacenters with inter-domain connectivity
- Via web-based service and management interface from any browser

Components

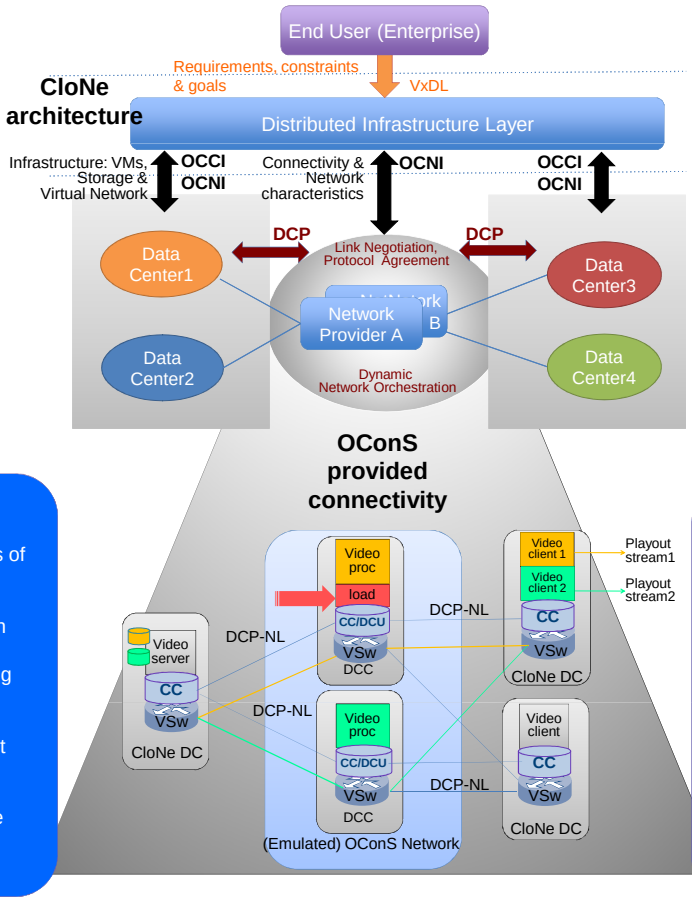
DCM: Distributed Cloud Manager (CloNe)

CC: Cloud Controller (CloNe)

DCU: Domain Control Unit (OConS DE)

DCC: Domain Control Client (OConS IE/EE)

DCP: Distributed Cloud Protocol



Interfaces
- OCNI-like interface to delegate and configure network connectivity
- DCP-Link Negotiation Protocol between attachment points of DC and network
- DCP-Network Level Protocol between OconS nodes and domains

Domain Control Unit (DCU)
- Combines concepts of
  - OpenFlow-based controller
  - Path Computation Entity (PCE) and
  - Traffic Engineering Database (TED)
- Realises OConS Orchestration Point (SOP)
- With Restful DCU interface for service invocation and orchestration

OConS Functions:
- Distributed (hierarchical or peer-to-peer) concept for cooperating controllers on end-to-end connectivity
- Managed paths for end-to-end flows realized by virtual switches
- Monitoring of network performance (nodes, links, flows)
- Redirection of application flows by VSwitch

Distributed video processing is used to demonstrate Elastic Networking between Cloud Nodes (data centers) and Network Domains

Seamless migration of video processing and application flows from one processing node to another based on load monitoring by DCP-NL

Cloud Network reacts on load changes in the processing node and on the links, even across multiple domains

Contact:
Michael Soellner[1], Peter Schefczik[1], Horst Roessler[1], Pedro Aranda Gutierrez[2]
[1]Alcatel-Lucent, [2] Telefonica

www.sail-project.eu

Figure A.1: OConS/CloNe Interaction, Poster at SAIL Booth, FuNeMS2012, July 2012, Berlin/DE

OConS in Action

- This demo aims at assessing the OConS orchestration feasibility by coordinating two mechanisms
  - Enhanced Access Selection (EAS): multi-criteria algorithm selection
  - Dynamic Mobility Anchoring (DMA): direct tunneling or dynamic activation of mobility anchors
- It focuses on OConS framework, showcasing
  - Registration of Entities & Mechanisms
  - Discovery of Entities
  - Smart Mechanism Management
- "Tailored" monitoring tool
  - Inter/Intra-node communication
  - OConS Signalling Messages

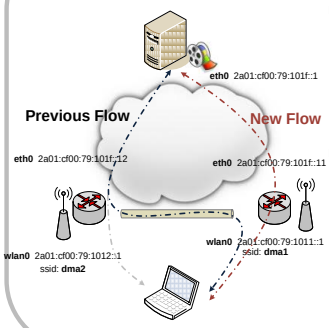
2013-02-13 SCALABLE & ADAPTIVE INTERNET SOLUTIONS

Figure A.2: OConS in Action: Orchestration, Brochure Introduction at Final Demonstration Event, Feb. 2013, Stockholm/SE

OConS in Action

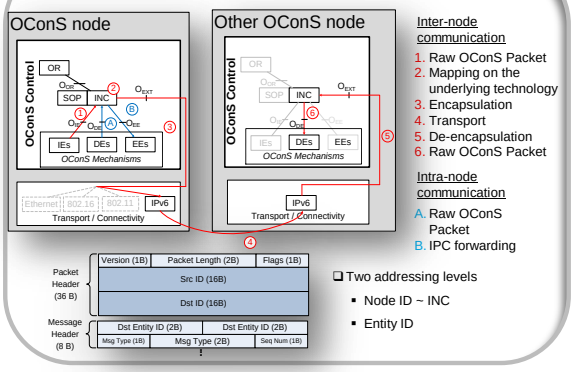
Orchestrating Enhanced Access Selection with Dynamic Mobility Anchoring mechanisms

Demonstrator Overview



- **Enhanced Access Selection (EAS) mechanism:**
  - Following the OConS architecture, mechanisms and protocols
  - Multi-criteria selection algorithm
- **Dynamic Mobility Anchoring (DMA) mechanism:**
  - Dynamic allocated mobility anchors in access routers
  - Direct IP routing of traffic flows (sessions) on mobile's current Access Router
  - Forwarding between the Access Routers

OConS-node Communication

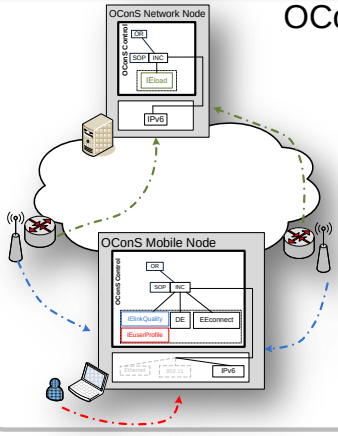


Inter-node communication
1. Raw OConS Packet
2. Mapping on the underlying technology
3. Encapsulation
4. Transport
5. De-encapsulation
6. Raw OConS Packet

Intra-node communication
A. Raw OConS Packet
B. IPC forwarding

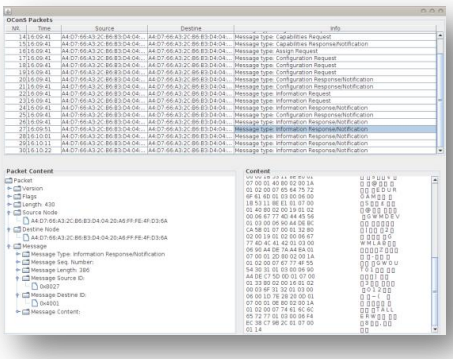
- Two addressing levels
  - Node ID ~ INC
  - Entity ID

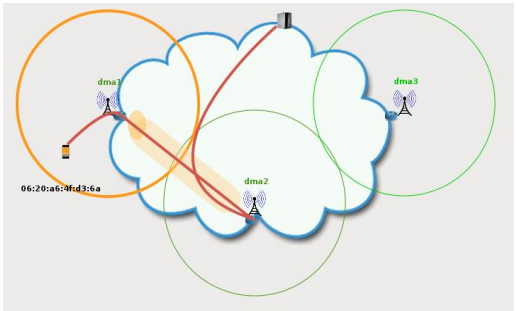
OConS Architecture Mapping



- **Tackle communications procedures**
  - Inter/Intra-entity communications
  - Message handling (packing, sending, receiving)
  - OConS signalling monitoring
- **Entities Bootstrapping**
  - Entities registration
  - Mechanism recognition and validation
  - Entities discovering
  - Dynamic mechanism(s) selection and adaptation

Monitoring and Visualization Tools





2013-02-13 SCALABLE & ADAPTIVE INTERNET SOLUTIONS

SEVENTH FRAMEWORK PROGRAMME

Figure A.3: OConS in Action: Orchestration, Poster at Final Demonstration Event, Feb. 2013, Stockholm/SE

OConS Multi-path Content Delivery with NetInf

Multi-path connectivity to networks in modern computing devices is becoming the norm in todays computing. These multiple paths can be used in many ways to benefit the users of these devices as well as the different service providers involved. The Information Centric Networking (ICN) architectures that are currently being defined such as Network of Information (NetInf), have considered the use of multiple paths natively. Though these architectures provide the use of multiple paths simultaneously, no formal mechanisms have been defined to utilise them in the best possible manner. The framework and the mechanisms defined in the Open Connectivity Services (OConS) work package of SAIL project identifies a number of multi-path strategies that utilise the multiple paths to request and receive content with NetInf in the best possible manner. The prototype developed and demonstrated in this work shows the operations of the OConS framework and OConS Multi-path NetInf (OMPNetInf) mechanism.

Demonstration Scenario

The demonstration is based on the Event with Large Crowd scenario developed in the SAIL project. The scenario consists of a flash crowd that requires downloading content based on spontaneous decisions that are related to the interests of the participants of the flash crowd. The participants use NetInf based computing devices and some of these devices are deployed with OMPNetInf. These devices are equipped with multi-path support and the operation of OMPNetInf demonstrate the use of multiple paths to deliver the required content.

Demonstration Sequence

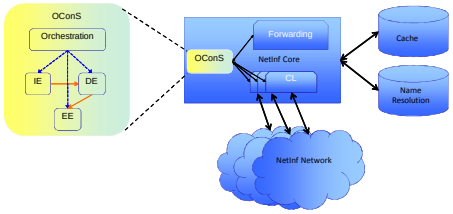
A participant (Bob) joins the flash crowd and starts downloading content. Other participants join the flash crowd and their activities result in congestion in some of the connected networks. This would usually result in Bob experiencing quality problems. But this is avoided due to the operations of OMPNetInf.

2013-02-13 SCALABLE & ADAPTIVE INTERNET SOLUTIONS

Figure A.4: OConS/NetInf Interaction, Brochure Introduction at Final Demonstration Event, Feb. 2013, Stockholm/SE

OConS Multi-path Content Delivery with NetInf

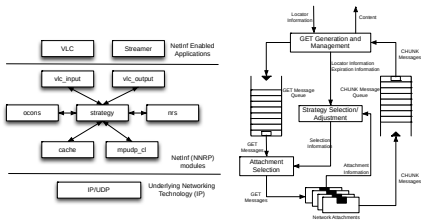
Architecture



NetInf Node Architecture Extended to Support Multi-path Content Delivery

- Open Connectivity Services (OConS) provides customized and optimized connectivity services for different types of networks, including Information Centric Networking architectures such as Network of Information (NetInf)
- OConS consist of a number of mechanisms that operate, utilizing the elements of the OConS Framework
- **OConS Multi-path Content Delivery for NetInf** (OMPNetInf) mechanism extends the NetInf architecture to provide efficient multi-path retrieval of content through the use of the OConS Framework
- OMPNetInf introduces a number of forwarding strategies to retrieve content:
  - **Splitting** – splits the content requests of one content stream into multiple paths
  - **Replication** – replicates the content requests of one content stream into multiple paths
  - **Distribution** – distributes the content requests of multiple content streams into multiple paths
- OMPNetInf utilizes the OConS Framework elements to select and use the appropriate multi-path strategy:
  - **Orchestration** – performs the discovery and selection of OConS mechanisms and the coordination of operations
  - **Information Elements** (IE) – collects and provides information to make multi-path strategy selection decisions
  - **Decision Elements** (DE) – performs the selection and configuration of forwarding strategy
  - **Execution Elements** (EE) – performs the enforcement of the decisions made by DEs

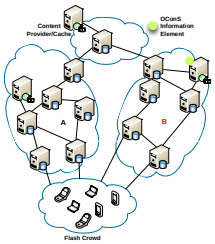
Prototype



NetInf Module Architecture and Splitting Strategy

- Based on NEC NetInf Router Platform (NNRP)
- Segmented content retrieval using NetInf GET and CHUNK (NDO) messages
- Transport overlaid on IP networks (UDP transport)
- Implements the Splitting Strategy
- Additive Increase/Multiplicative Decrease based path utilization
- Deployed in user devices
- Uses the OConS Framework elements to make decisions on how best to use multiple paths
- Module functionality:
  - **vlc input** – handles the requests for content from NetInf enabled Video LAN Client (VLC) applications
  - **vlc output** – handles the serving of content requests of NetInf enabled applications
  - **strategy** – handles the Orchestration and DE functionality of the OConS framework to select and configure the multi-path strategy
  - **ocons** – handles the interactions with the IEs to obtain information
  - **nrs** – handles the name resolution functionality
  - **mpudp cl** – handles the EE functionality of OConS where the selected strategy is implemented
  - **cache** – handles the caching of content

Demonstration: Flash Crowd Participants using OMPNetInf

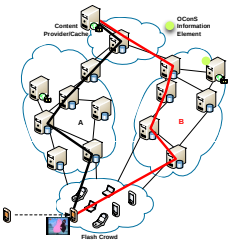


Scenario Details

- A flash crowd builds up to see an event discussed and initiated over a social network
- Participants are interested in downloading and watching a related video

Technical Details

- Participants carry NetInf enabled computing devices
- Participants use NetInf networks in the vicinity to download content

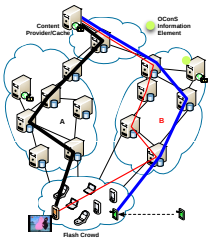


Scenario Details

- Bob joins the flash crowd
- He starts to download a video

Technical Details

- OMPNetInf mechanism utilizes the Splitting Strategy to distribute the content retrieval over multiple paths
- Information provided by OConS is used to determine the splitting ratios



Scenario Details

- Other participants join the flash crowd and starts to download content
- Bob should usually experience problems with video quality but the quality remains as expected

Technical Details

- Congestion occurs due to loaded networks
- OConS provide information on congestion in connected networks
- Splitting Strategy re-adjusts the content retrieval distribution ratios to minimize the use of congested networks

Asanga Udugama[1], Andreas Timm-Giel[2], Sameera Palipana[1] and Carmelita Görg[1]

[1] University of Bremen, Bremen, Germany
[adu|cg|yzaki]@comnets.uni-bremen.de

[2] Hamburg University of Technology, Hamburg, Germany
timm-giel@tuhh.de

www.sail-project.eu

Figure A.5: OConS/NetInf Interaction, Poster at Final Demonstration Event, Feb. 2013, Stockholm/SE

List of Acronyms

3G	Third Generation
AJAX	Asynchronous JavaScript and XML
API	Application Programming Interface
AR	Access Router
ASF	Advertising Supporting Function
BPQ	Bundle Protocol Query
CL	Convergence Layer
CloNe	Cloud Networking
CSV	Comma-Separated Values
CUC	Cloud Unit Controller
DC	Datacentre
DCC	Domain Control Client
DCC-B	Domain Control Client - Border Forwarding Node
DCC-I	Domain Control Client - Interior Forwarding Node
DCM	Distributed Cloud Manager
DCP-LNP	Distributed Cloud Plane - Link Negotiation Protocol
DCU	Domain Control Unit
DCUP	Domain Control Protocol
DE	Decision Making Entity
DFC	Dynamic Flow Control
DMZ	De-militarized Zone
DS	Data Source
DTN	Delay Tolerant Network
EA	Execution Accomplishment
EE	Execution and Enforcement Entity
ETSI	European Technical Standards Institute
EwLC	Event with Large Crowd
FNS	Flash Network Slice
GUI	Graphical User Interface
HURRy	HUman Routines optimize Routing
ICN	Information-Centric Networks

IDE	Integrated Development Environment
IE	Information Management Entity
INC	Inter/Intra-Node Communication
IPv6	Internet Protocol Version 6
IPC	Inter Process Communication
ISI	Infrastructure Service Interface
JDK	Java Development Kit
JSON	JavaScript Object Notation
KVM	Kernel-based Virtual Machine
MN	Mobile Node
NDO	Named Data Object
NE	Network Element
NetInf	Network of Information
NfV	Network Function Virtualization
NIC	Network Interface Card
NNRP	NEC NetInf Router Platform
OCNI	Open Cloud Networking Interface
OConS	Open Connectivity Service
OF	OpenFlow
OMPNetinf	OConS Multi-path Network of Information
ONF	Open Networking Foundation
OR	Orchestration Register
OSAP	Orchestration Service Access Point
OSGi	Open Service Gateway Initiative
OVS	Open Virtual Switch
PCE	Path Computation Entity
PRoPHET	Probabilistic Routing Protocol based on Historical EncounTers
pyOCNI	Python Open Cloud Networking Interface
QEMU	Quick EMUlator
QoE	Quality of Experience
QoS	Quality of Service
REST	Representational State Transfer
RIB	Routing Information Database
RPC	Remote Procedure Call
RPE	Request Processing Entity
RSSI	Received Signal Strength Indicator
SAIL	Scalable Adaptive Internet Solutions

SDN	Software Defined Networking
SNR	Signal to Noise Ratio
SOA	Service-Oriented Architecture
SOAP	Simple Object Access Protocol
SOP	Service Orchestration Process
TED	Traffic Engineering Database
TLV	Type Length Value
VM	Virtual Machine
WAN	Wide Area Network
WP	work package
XML	Extensible Markup Language

List of Figures

List of Tables

References

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

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

[3] SAIL. Demonstrator Specification and Integration Plan. Deliverable FP7-ICT-2009-5-257448-SAIL/D.C.3, SAIL project, May 2012. available online from http://www.sail-project.eu.

[4] SAIL. Architectural Concepts of Connectivity Services - Addendum. Deliverable FP7-ICT-2009-5-257448-SAIL/D.C.1 Addendum, SAIL project, January 2012. Available online from http://www.sail-project.eu.

[5] SAIL. Architecture and Mechanisms for Connectivity Services. Deliverable FP7-ICT-2009-5-257448-SAIL/D.C.2, SAIL project, July 2012. Available online from http://www.sail-project.eu.

[6] SAIL. Description of Project-wide Scenarios and Use Cases. Deliverable FP7-ICT-2009-5-257448-SAIL/D.A.2, SAIL project, February 2011. Available online from http://www.sail-project.eu.

[7] SAIL. Applications for Connectivity Services and Evaluation. Deliverable FP7-ICT-2009-5-257448-SAIL/D.C.4, SAIL project, May 2013. available online from http://www.sail-project.eu.

[8] Luis Diez, Olivier Mehani, Lucian Suciu, and Ramón Agüero. Design and implementation of the open connectivity services frameqork. In *MONAMI 2012, 4th International Conference on Mobile Networks and Management*, 2012.

[9] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1994.

[10] Java Unix Domain Sockets. `http://code.google.com/p/juds/`.

[11] SAIL. Final Harmonised SAIL Architecture. Deliverable FP7-ICT-2009-5-257448-SAIL/D.A.2, SAIL project, February 2013. Available online from http://www.sail-project.eu.

[12] VirtualBox General-purpose Virtualizer . `http://www.virtualbox.org/`.

[13] Bob Lantz, Brandon Heller, and Nick McKeown. A network in a laptop : Rapid prototyping for software-defined networks. *Electrical Engineering*, pages 1–6, 2010.

[14] SAIL. Architectural Concepts of Connectivity Services. Deliverable FP7-ICT-2009-5-257448-SAIL/D.C.1, SAIL project, July 2011. Available online from http://www.sail-project.eu.

[15] 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.

[16] Fariborz Derakhshan, Heidrun Grob-Lipski, Horst Roessler, Peter Schefczik, and Michael Soellner. On converged multidomain management of connectivity in heterogeneous networks. *Future Network and Mobile Summit (FuNeMS2012), Berlin.*, July 2012.

[17] Jeff McAffer and Jean-Michel Lemieux. Eclipse rich client platform: Designing, coding, and packaging java applications, 2005.

[18] Beacon OpenFlow controller. `http://www.openflowhub.org/display/Beacon/`. Last seen on Wed, 09 May 2012.

[19] A. Lindgren, A. Doria, E. Davies, and S. Grasic. Probabilistic Routing Protocol for Intermittently Connected Networks. RFC 6693(Experimental), August 2012.

[20] J. M. Cabero, V. Molina, I. Urteaga, F. Liberal, and J. L. Martin. Acquisition of human traces with bluetooth technology: Challenges and proposals. *Ad Hoc Networks*, Accepted. To appear.

[21] Communication System Design. Android application version 3 of the project bytewalla, 2010.

[22] Telecom Business Area. Android implementation of hurry routing protocol and bpq extension block for dtn2 taking the bytewalla project as basis, 2013.

[23] S. Farrell, A. Lynch, D. Kutscher, and A. Lindgren. Bundle protocol query extension block. Internet Draft draft-farrell-dtnrg-bpq-01.txt, Work in progress, March 2012.