Deliverable 3.5: Programmability framework prototype report

Dissemination level	RE
Version	0.1
Due date	31.07.2016
Version date	31.07.2016

This project is co-funded
by the European Union

Document information

Editors

Balázs Sonkoly (BME)

Contributors

Pontus Sköldström (ACREO), János Czentye (BME), Balázs Németh (BME), Attila Csoma (BME), Dávid Szabó (BME), Balázs Sonkoly (BME), Jokin Garay (EHU), Dávid Jocha (ETH), Raphael Vicente Rosa (ETH), Róbert Szabó (ETH), Sahel Sahhaf (iMinds), Wouter Tavernier (iMinds), Steven Van Rossem (iMinds)

Coordinator

Dr. András Császár
Ericsson Magyarország Kommunikációs Rendszerek Kft. (ETH)
Könyves Kálmán körút 11/B épület
1097 Budapest, Hungary
Fax: +36 (1) 437-7467
Email: andras.csaszar@ericsson.com

Project funding

7th Framework Programme
FP7-ICT-2013-11
Collaborative project
Grant Agreement No. 619609

Executive Summary

This deliverable provides the final report on the prototypes developed by WP3. The main outcome of WP3 is the programmability framework for the UNIFY architecture [D2.2] encompassing all components designed, implemented and evaluated during the project. The initial version of the framework was delivered in [D3.1], while the key processes, specific functionalities and required algorithms were identified in [D3.2]. In addition, two alternative Network Function Forwarding Graph (NF-FG) representations and corresponding process flows were investigated and specified in [D3.2a]. These works resulted in the revised version of the programmability framework delivered in [D3.3], and in the main WP3 prototype provided by [D3.4]. Therefore, this prototype report is specific to the latest revisions of the framework and to the integration efforts. We assume that the reader is familiar with the content of the predecessor documents.

On the one hand, this prototype report includes modifications and updates which were required during the final integration of the software components provided by WP3. More exactly, $i)$ we have addressed the integration of additional functionalities with ESCAPE (our main prototyping framework), $ii)$ we have provided further performance evaluation on the framework, $iii)$ we have extended our Virtualizer-based NF-FG model and implemented the updates in the new version of the Virtualizer library (Virtualizer v5).

On the other hand, this deliverable focuses on the use cases highlighted in [D3.3]. More specifically, the WP3 related parts of the Elastic Router and FlowSNAC (combination of FlowNAC and state migration) use cases have been implemented and integrated in working prototypes. These integration activities required the cooperation with WP4 and WP5 participants in order to make use of Service Provider DevOps (SP-DevOps) observability features and to support the inter-operation between the programmability framework and the Universal Node (UN) domains, respectively. Moreover, this work and cooperation provided essential steps towards the Integrated Prototype of UNIFY.

With the reported updates we arrived to a fully functional joint cloud and network domain abstraction and programmability framework, which is capable of automated service function chain deployment over multiple domains irrespective of their applied technologies. Besides the generic resource orchestration framework we implemented and analyzed elasticity control and state migration in details for the purpose of illustrating capabilities and usage of the framework.

With the current prototypes we achieved our goals and objectives to design and implement automated service chaining for multi-domain and multi-technology environments in a fully flexible and programmatic way. The design reports are publicly available, relevant parts were submitted to standardization bodies and the core components (Virtualizer, ESCAPE, Universal Node, ...) were released open-source.

Known limitations of the designs and prototypes are the handling of multiple administrations and the handling of domain dynamism (appearance and disappearance of a domain). Both of these are addressed in a sequel project of Horizon 2020, 5G-PPP, 5G Exchange, which builds on the latest achievements and prototypes of UNIFY. Therefore, we foresee further extensions and updates to the various components.

Last but not least the WP3 prototype components build the core of the project's integrated prototype.

Contents

List of Figures

1 Introduction

This deliverable together with [D3.3] and [D3.4] provides a final report on the UNIFY programmability framework designed, implemented and evaluated by WP3.

The rest of the document is structured as follows. Section 2 presents recent results and activities related to our orchestration framework called ESCAPE[1]. More specifically, Section 2.1 summarizes the relevant modifications and updates on the framework which were required for integrating several software components provided by WP3. Section 2.2 provides a performance evaluation study on the online embedding algorithm which is the key element of the resource orchestration process. In Section 2.3, we provide a framework proposal (in compliance with the orchestration framework) for VNF benchmarking which enables an orchestrator to have a coherent understanding of performance vs. resource needs for VNFs specific to execution environments.

Section 3 deals with various aspects of domain virtualization. In Section 3.1, we summarize the recent extensions and updates of our Virtualizer library. The current version is Virtualizer v5. Section 3.2 addresses the construction of the virtualizers, i.e., the exposed virtual resource views towards upper level resource orchestrators.

Section 4 is devoted to the use cases and related efforts towards the Integrated Prototype of UNIFY. The Elastic Router use case, the required processes and implementation issues are discussed in Section 4.1. The integration of FlowNAC and the state migration process is presented in Section 4.2. This use case is referred to as FlowSNAC. A novel embedding algorithm taking decomposition choices into account is introduced in Section 4.3. The details of the integration with ESCAPE are also given showing the benefits of our extensible, modular framework.

Finally, Section 5 concludes the deliverable and the activities of WP3.

Besides the main body of the document, Appendix A provides further details on the multi-domain, multi-technology aspects of our orchestration framework.

[1] Extensible Service ChAin Prototyping Environment using Mininet, Click, NETCONF and POX (ESCAPE)

2 ESCAPE orchestration framework

2.1 ESCAPE updates

ESCAPE is our main proof of concept prototyping framework of WP3 which was introduced first in [D3.2] and further described in more details in [D3.4]. ESCAPE has been improved and extended with new features and components to enable the integration of tools from other partners and fully utilize the new features developed in the meantime. This section is devoted to summarize and give a detailed description of the main changes and progress of ESCAPE compared to the previous state in [D3.4]. These changes are summarized in the following subsections which are organized according to the affected ESCAPE modules and components.

2.1.1 Changes related to REST-APIs

The REST-APIs of ESCAPE orchestrator have been updated to increase performance and adapt the new version of Virtualizer library, namely the Virtualizer v5 (see Section 3.1).

Virtualizer v5. The ROS API, which realizes the Sl-Or interface of the UNIFY architecture and the CfOr API, which refers to the Cf-Or interface respectively, use a centralized converter class called NFFGConverter to convert the requested topology descriptions and the received service requests back and forth in a transparent way. This new class can convert between the XML-based Virtualizer format and ESCAPE's internal NFFG format independently and without 3rd party packages in order to be able to be distributed along with our NFFG library. The conversion methods are reconstructed and separated in the granularity level of the base NFFG entities such as Big Switch and Big Software Nodes (BiS-BiS Nodes) nodes, Network Functions (NFs), flowrules, etc. to ensure verifiability and easy adaptation of future Virtualizer versions.

The conversion mechanism utilizes the new attributes introduced in Virtualizer v5 to expand the content and reduce the size of the output format:

- Use new port fields to specify SAP characteristics (see Section 3.1.1).
- Use new resource fields in SAP port to provide necessary resource information for inter-domain connections created by ESCAPE's domain merging algorithm.
- Use metadata field in Node to transfer additional resource data (delay, bandwidth) of Infrastructure Node instead of intra-node links (see Section 3.1.5).
- Extend internal NFFG model with metadata fields corresponding to metadata fields in Virtualizer v5.
- Use metadata field in Virtualizer container to transfer end-to-end requirement links between ESCAPE instances. In this case local ESCAPE can use the domain-wide requirements calculated by global ESCAPE for better orchestration than best effort.
- Enforce relative paths explicitly to shrink link and port references and avoid inaccurate difference calculation.

The REST-APIs have been extended to be able to receive a difference-based domain view along with the full view as a valid service request. The difference calculation and re-patching fully rely on the high-level functions of the Virtualizer library such as diff(), patch() and reduce() therefore this feature is only supported in case the API is configured as a UNIFY interface.

The diff-based communication requires a high level of consistency considering the ID of regenerated flowrules, the unique ID of nodes and dynamically created ports. To ensure the consistency of generated IDs the flowrule conversion has been enhanced to use the ID of the predecessor service hop as generation seed instead of a simple counter. With this approach the ID of analogous flowrules remains the same during iterative service requests.

The REST-APIs can also be configured in the global configuration file to use the internal NFFG format for backward compatibility. For this reason REST functions strongly depend on the standard Content-Type header of the HTTP protocol to indicate the generated output format or to determine the used format of a received request. If the expected header field is not given, the APIs try to recognize the format based on the standard headers and declarations as a last resort to support wide range of domain agents.

Resource view caching. The REST-API methods which respond the domain view (the ordinary get-config function of the Sl-Or interface) implement a cache functionality by returning a stored resource view in case the domain had not changed since the last request. The change recognition has been integrated into the core mechanism of our Virtualizer/Filter objects which are constantly bound to the specific handler entities (for further description, see the next paragraph). The API handlers only store the fully prepared output structure of the domain view to eliminate the unnecessary conversion of an unchanged topology if the API is configured as UNIFY interface.

Due to the elimination of the aforementioned CPU consuming steps, ESCAPE provides better performance and lower response time for resource requests, which makes it more suitable to use as a local orchestrator.

Improved filtering. The resource requesting mechanism has been reworked in the main modules referred to the Service Layer (SL) and Orchestration Layer (OL) in order to integrate our filtering entities called Virtualizers in a more transparent way (should not be confused with the Virtualizer library, which among other things provides the transport format of Sl-Or). Every resource view request directly passes through an assigned Virtualizer object by default. These Virtualizer entities ensure the implicit virtualization of network elements which introduced in [D2.2]. The REST-APIs and ESCAPE's internal Resource Orchestrator (RO) module have extended to allocate specific Virtualizer objects at initial time to realize the virtualized views for the Sl-Or interfaces. Similarly, the CfOr API has been extended for the Cf-Or interface as well. The type of virtualization can be set in the configuration file both for the REST-APIs and for the internal Sl-Or interface located between ESCAPE's SL and OR layer modules.

The SingleBiSBiS view generation has been enhanced to give more accurate information about the virtualized topology. The main updates are the following:

- Virtualized Network Functions (VNFs) and flowrules are included in the Single BiS-BiS node (SBB) generation, hence it can be used on any mapped topology, not only simple resource graphs.
- Resource value aggregation is refined to give a better overestimation and to avoid false negative mapping results.
- The SBB node has constant ID and name now. This naming convention allows to calculate the correct difference for the Virtualizer library.
- Flowrule virtualization is added to the SBB node by creating overarching flowrules which refer to the original Service Graph (SG) hop links. The flowrule recreation is based on a new field (hop_id) of the Flowrule class in our NFFG format. The hop_id is mainly used and set by the mapping algorithm for backtracking predecessor SG hop IDs.

2.1.2 Changes in the orchestration process of service requests

Along with the development of the core mapping algorithm, the pre and post orchestration tasks have been extended in ESCAPE to meet the new requirements and to provide the requested features came from other partners and work packages.

Extended preprocessing. The preprocessor component of the core mapping algorithm has been extended with additional input checks to be able to handle deficient service requests. The mapping algorithm can now map a request without E2E (end-to-end) Requirement links or explicit SG hop links. The mitigation of input requirements extends the mapping algorithm to be fully feasible for a local domain orchestrator in which case the service requests are mostly received in Virtualizer format through one of the Sl-Or interfaces.

The Virtualizer model has no dedicated field for requirement links of our NFFG format, but the conversion has been extended to transfer these requirements in the newly introduced metadata field. Moreover, the core algorithm has been improved to split the E2E requirement links on an Infrastructure Node basis and assign the fragments to the involved BiS-BiS nodes on the limited end-to-end path. This requirement partition step enables ESCAPE to handle non-SBB resource graph, which consists of multiple domain views, and to provide complementary information to local domain orchestrators. Without the E2E requirements, the updated core algorithm performs the mapping on a best-effort approach instead of cancelling the mapping with an error.

If no SG hop is given in the service request, the mapping preprocessor tries to restore the original SG hop links based on the given flowrules. The assignment between a generated flowrules and the predecessor SG hop is identified with the help of the hop_id field which must be set in a previous mapping step by an upper level orchestrator. If there are missing flowrules or misconfigured hop references in the request, the preprocessor aborts the mapping process with a specific exception (BadInputException) and in that case ESCAPE also refuses the given service request.

The domain resetting feature, which provide the capability to restore the first received state of the registered domains, has been reconsidered from the perspective of local orchestration. ESCAPE running as a local orchestrator may receive a completely empty resource graph (which had been most likely reported by itself). In order to avoid the mapping running into an error the request validation has been extended to detect empty requests (without any NF or flowrule) in ESCAPE's Orchestrator classes. In this case the Orchestrator classes skip the mapping process and forward the empty request as the result to the lower layers. As the final step the traffic steering task automatically manages the changes in the flowrule set. If the request was a clean-up message with a completely empty NF-FG, the installed NFs and flowrules will be removed from the Mininet-based infrastructure due to the basic behaviour of ESCAPE's Controller Adaptation module.

Pre-mapping validation. A new validation feature has been implemented and integrated into the core orchestration process of the SL and OL modules. The pre/post validation enables external tools to easily inject validation actions and abort the whole orchestration process on demand. There are two ways to integrate an NFFG validation tool into ESCAPE:

1. Create a standalone class which implements the AbstractMappingDataProcessor abstract class and overrides the necessary function(s) according to the type of the desired injection point(s). These predefined functions will be invoked in the dedicated point of the orchestration process automatically if the validation is enabled for the relevant layer in the global configuration file (under the PROCESS name). Currently the following functions are defined with the injection points:

    - pre_mapping_exec(request, resource) is invoked immediately before the core mapping process.

- post_mapping_exec(request, resource, result) is invoked immediately after the mapping process with the mapping result if no exception was raised during the mapping.

There is also multiple way to abort the orchestration process: i) return with a value considering True in Python or ii) directly raise the dedicated ProcessorError exception.

2. Subscribe to the specific PreMapEvent event to invoke the proper processing tasks directly before the mapping process and to the PostMapEvent event for the post-mapping tasks, respectively. These evens are published by the core orchestrator functionality in AbstractOrchestrator abstract class by default.

An optional visualization feature has been integrated into the orchestration process, and as a result, ESCAPE is able to send the different stages of the orchestration process (resource and request graphs) to a remote server in Virtualizer format. The server parameters can be set in the global configuration file under the dedicated top-level entry visualization. To activate the remote visualization the escape.py startup script need to be started with a special initial flag (-V).

The inter-module event communication has been reconsidered, and now the overall result of an installation process can be propagated to the upper layers with details about the success.

2.1.3 Changes in the Controller Adaptation module

The installation process, which is realized by the Controller Adapter (CA) module of ESCAPE, has been enhanced to support and complete the latest changes of the upper layer modules and to solve the challenges came from the expanding multi-domain architecture.

Improved adaptation process. The main adaptation process has been restructured with respect to the installation steps of the registered domains. The partial result of a domain installation is handled in a more separated and independent way so that the final result can be propagated upwards with more accurate details about the process. This approach allows the Adaptation module to continue the whole NFFG installation despite the fact that one (or more) of the individual installation steps was unsuccessful. The separated domain handling also provides the way to manage the state of Domain Virtualizer (DoV) by domains.

The respective Adapter classes communicating with the domain agents through REST-APIs have been modified to support the separated result handling. The RPC wrapper functions are altered to report back the result status of the requests. Additionally, in these session-based Adapters a timeout event does not cause the installation process to report a failed attempt in order to support different domain agents with serialized process queue or with single-threaded implementation.

The domain splitting and merging functionalities (along with other functions such as SBB generation) have been generalized and moved to a central container class, namely the NFFGToolBox to make these features portable, reusable and easily tested with different testing methods such as Unit tests. These functions have also been extended to utilize the new fields in Virtualizer v5 in accordance with the improved conversion class.

To support the E2E requirement splitting in the mapping algorithm, the main adaptation process has been extended with an E2E rebind step. The rebinding function searches for requirement fragments in SBB views and recreates the related requirement links as SAP-SAP connections.

Extended DoV management. Due to the better support of the autonomous domains, the DoV management has been extended with basic steps in terms of domain-wide operations. The domain addition, deletion and update, where all input is in the internal NFFG format, have been generalized and implemented in the NFFGToolBox class to grant the basic tool set for:

- DoV construction during initiation of enabled DomainManagers,
- domain-based DoV update during NFFG installation,
- on-the-fly reaction of domain detection/loss,
- and polling-based domain management.

The Virtualizer-NFFG conversion has been upgraded with a unique ID generation for the Infrastructure Nodes in order to support the collision-free merging of autonomous domains and avoid duplicated names or IDs. The unique ID generation is realized by extending the node ID with the unique domain name. These extended IDs are only used internally and the forwarded requests contains the original IDs.

2.1.4 Changes in domain management

The domain management entities have been significantly redesigned in the CA module to increase re-usability and to facilitate the configuration of ESCAPE.

Restructured Manager/Adapter hierarchy. The DomainManager classes have been completely restructured to use dedicated DomainAdapter classes, in which case the Adapters are instantiated and used only by the container Manager instance, instead of using only one global Adapter instance. This hierarchical structure allows equivalent Manager classes, which are defined to different domains, to reuse the same Adapter class with different configuration. Moreover, Managers may be configured with different Adapter implementations transparently if the role of these Adapters (and consequently the interfaces) is identical. The Adapter roles are defined on the basis of the access mechanism and communication format of the supervised domain part. The reusable Adapter and Manager classes enable ESCAPE to manage multiple domains of the same type and distinct domains with a common form of communication. From the perspective of UNIFY these enhancements upgrade ESCAPE to be able to orchestrate multiple OpenStack (OS), Universal Node (UN), Docker, etc. domains all at once and/or different domains using the common Virtualizer format.

The global configuration file has also been significantly restructured to comply with the hierarchical relation between the Manager and Adapter classes. The Adapters assigned to an overseer Manager have to be given in the Manager's configuration section under the `adapters` name. The default configuration of `InternalDomainManager` in Listing 1 demonstrates the hierarchical structure of Manager-Adapter entities.

A new and reusable Manager and Adapter class have been implemented for the universal management of UNIFY domains. The `UnifyDomainManager` class contains the common top-level functionality and the `UnifyRESTAdapter` is responsible for the communication with the remote domain agents over HTTP in the Virtualizer format. The Adapter class uses the central conversion class for format conversion and relies on the standard RPC names of the Sl-Or interface (`get-config`, `edit-config`, `ping`). The wrapper functions support the difference calculation operation to be able to send only the minimal change set as a service request. If the difference calculation is enabled, the service install step (`edit-config`) is automatically extended with an additional topology request (`get-config`) for the most recent state of the domain to be able to calculate the accurate difference. The dedicated domain Managers are preserved for backward compatibility and for unexpected alterations but due to the unified domain management they have been modified along with the `RemoteESCAPEv2RESTManager` to inherit from the common `UnifyDomainManager`.

Listing 1: Default configuration of InternalDomainManager: JSON view

```
{
  "INTERNAL": {
    "module": "escape.adapt.managers",
    "class": "InternalDomainManager",
    "poll": false,
    "adapters": {
      "CONTROLLER": {
        "module": "escape.adapt.adapters",
        "class": "InternalPOXAdapter",
        "name": null,
        "address": "127.0.0.1",
        "port": 6653,
        "keepalive": false
      },
      "TOPOLOGY": {
        "module": "escape.adapt.adapters",
        "class": "InternalMininetAdapter",
        "net": null
      },
      "MANAGEMENT": {
        "module": "escape.adapt.adapters",
        "class": "VNFStarterAdapter",
        "username": "mininet",
        "password": "mininet",
        "server": "127.0.0.1",
        "port": 830,
        "timeout": 5
      }
    }
  }
}
```

Improved polling. The common polling mechanism in the core AbstractDomainManager class has been extended and enhanced to be fully adaptable to the local domain agents developed in the project. If the polling is configured for a DomainManager, the view of the managed domain in DoV is maintained separately with the help of the primary DoV operations (described in 2.1.3). This means that during the installation of a service request the DoV will not be updated immediately if polling is enabled. In this case the result of the local service installation will be propagated into DoV by the periodic get-config requests.

The topology processing has been improved with the capability of change detection, which relies on the reduce feature of the Virtualizer library. The change detection completed with the caching functionality of the REST-APIs (see section 2.1.1) gives a better performance for local ESCAPE by eliminating the unnecessary update steps (along with the unnecessary conversations or SBB creation).

Local orchestration enhancements. The Infrastructure Layer (IL) module of ESCAPE has also been improved to meet the requirements of local orchestration. The instantiation of the click-based NF has been extended with an all-clearing step for supporting NF removal. With this approach, as in the case of flowrule injection, every service installation is started with the removal of all initiated NF. Currently this complete domain clearing approach is the most applicable solution for iterative service installation considering that the core mapping algorithm provides the fully remapped topology and the Mininet-based infrastructure does not support the NF migrations between our internal Execution Environments.

Furthermore, the NF/flowrule deletion implicitly provides the basis for the domain resetting feature and decremental service requests.

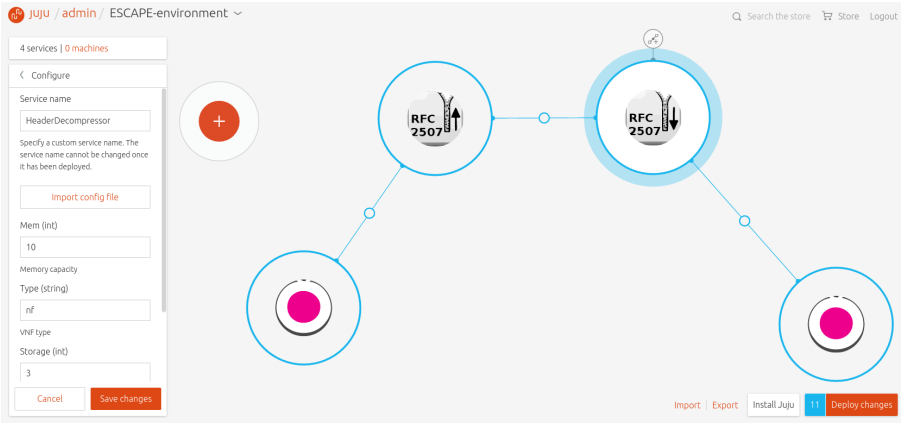


Figure 1: GUI main menu with some VNF

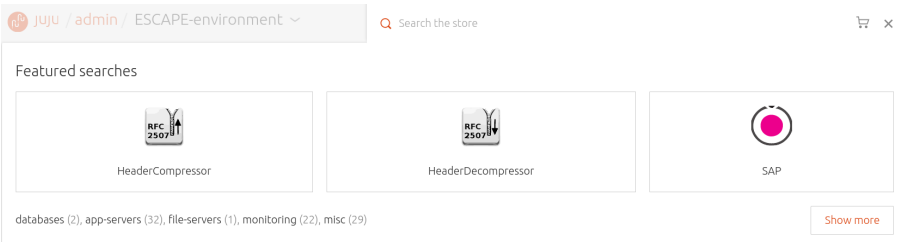


Figure 2: Menu of the VNF Store

2.1.5 Integration of Juju GUI with ESCAPE

At the highest level, ESCAPE uses Juju GUI which is a JavaScript and HTML based web application for managing and monitoring services in a user friendly way. Juju GUI is originally created for Juju, that is a "state-of-the-art, open source, universal model for service oriented architecture and service oriented deployments". Due to the similar functionality of ESCAPE and Juju, Juju GUI is a convenient choice to be used and fine-tuned for ESCAPE as the default GUI.

With Juju GUI service definition becomes a simple three-step procedure:

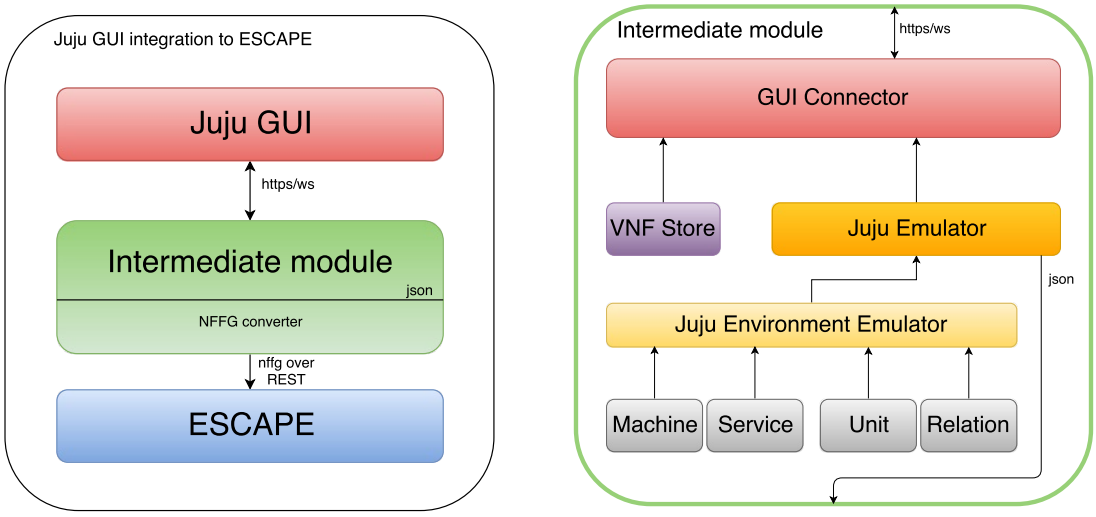
1. deploy: in the top navigation menu (Figure 1) we can explore the available VNFs (Figure 2) and use them to create any service graph. The list of available VNFs can be loaded easily from a custom VNF Store.

2. configure: any VNF or relationship can be fine-tuned by filling the options in its separate configuration panel displayed at the left (Figure 1).

3. expose service: after the service graph is created with a final confirmation it is sent to ESCAPE through the REST API.

After the service is started Juju GUI provides real-time information about the installed VNFs by sending queries periodically through a websocket (ws). Since Juju GUI originally is prepared to talk with Juju core an intermediate module is created between Juju GUI and ESCAPE (Figure 3a) that emulates Juju core and translates the messages appropriately.

This design choice has the advantage that only minimal modification is required in the original Juju GUI codebase and enables us the continuous upgrade of the GUI as newer versions come out. To communicate with ESCAPE the intermediate module contains an NFFG converter to create proper NFFG structures from the VNF chains. Communications between the intermediate module and NFFG converter taking place with json over http to achive maximal flexibility (this enables running these modules on different servers if it is required).



(a) GUI integration with ESCAPE

(b) JujuEmulator details

Figure 3: Details of ESCAPE and Juju GUI integration

The intermediate module (Figure 3b) emulates a simple Juju core with adaptation to ESCAPE requirements. To achive this it creates a webserver and waits for websocket and https connections on predefined ports.

GUI Connector: the main module that handles Juju GUI messages. Since Juju GUI communicates through multiple protocols (like https and websocket) it serves as a router sending messages toward the proper sub-modules.

VNF Store: the https queries for available VNFs are routed here and the VNF Store answers with the list of implemented VNFs. Currently information about VNFs are stored in a file system, but it can be easily turned into a database by defining the proper handlers.

Juju Emulator: it handles requests, sent by websocket queries, for status and update information about the VNFs. Juju GUI is able to create/delete/update links (relationships) and nodes (VNFs) in the service graph through communication with the JujuEmulator.

Juju Environment Emulator: it maintains the state of the service graph through node types (elements) listed below. Every quires and updates are executed in this module. Name convention is inherited from the original Juju name convention to ease code maintanace, typical examples are Machine, Service, Unit, Relation.

Machine: gives information about the real devices on which the VNFs are running.

Service: after an abstract VNF created with a specified configuration and requirement set in Juju Environment Emulator it creates a Service element.

Unit: describes the service and the machine binded together with status information (like alive, stooped, pending etc.).

Relation: describes the relationships between services (VNFs) with their metadata as well.

2.1.6 Other miscellaneous modifications

ESCAPE has several additional modification which does not belong to any layer module. The following list summarizes these major additions:

- Install script (`install-dep.sh`) is restructured and decomposed to provide the possibility to install any functional part of ESCAPE (such as Global Orchestrator, Local Orchestrator and our experimental GUI) separately.

- Experimental NetworkX based GUI is integrated with the startup script (escape.py) for debugging (-g). This simple GUI component can render internal NFFG structures and can be useful for debugging purposes.
- New logging level is introduced for more verbose logging (-vv) in which case the sent, received, calculated and modified data structures are logged to the standard output.
- Many error handling and format checking are added to improve robustness of ESCAPE.

2.2 Performance evaluation of the embedding process

One of the key elements of the resource orchestration is the optimization algorithm which is responsible for mapping requests to available resources. In our case, service graphs (containing virtual network functions and logical links between them) have to be embedded in substrate graphs (describing physical or virtual infrastructure), however, the requirements can be very tricky (e.g., delay requirements between arbitrary network functions, end-to-end bandwidth requirements, collocation constraint on given network functions). There is a number of algorithms proposed for similar or related problems following two different approaches. On the one hand, offline algorithms receive all requests as an input and seek for optimal or close to optimal solutions typically at the expense of long runtimes. On the other hand, online algorithms process individual requests and provide solutions on the fly based on heuristics.

We have proposed and implemented a novel customizable algorithm for service graph embedding which is able to jointly control and optimize the usage of compute and network resources, and it inherently supports network function sharing between multiple requests. The algorithm has also been integrated with our ESCAPE orchestration framework.

2.2.1 Problem Definition

The problem of Virtual Network Embedding (VNE) is well-studied by the researchers of network virtualization. In VNE, the goal is to allocate network and computation resources for the requested virtual networks on a shared substrate network. Virtual nodes shall be hosted on substrate nodes, and communication links between virtual nodes shall be mapped to paths in the substrate network. An extensive survey on VNE algorithms collects the common elements from VNE problem variants and provides a detailed overview on the state-of-the-art solutions [Fis+13]. Most of the variants deal with constraints on the virtual nodes and links, such as requested number of CPU-s and minimal bandwidth, respectively. An example formal definition for the VNE problem can be found in [CRB]. Service Function Chaining (SFC) has been revitalized based on the recent achievements of SDN and NVF researches. As a result, a novel mathematical problem has arisen rooted in VNE which we refer to as Service Graph Embedding (SGE).

Table 1: Mathematical notations used in this section.

Notation	Description
$\mathrm{V}(G), \mathrm{E}(G)$	Vertices and edges of G
$\mathrm{P}(G)$	All simple paths of G
$\mathrm{P}_S(G)$	Simple paths of G starting and ending in SAPs
$\Psi(SG) = \{c_p \vert p \in \mathrm{P}_S(SG)\}$	Set of E2E chains
$c_p = (B_p, D_p, p)$, $B_p, D_p \in \mathbb{R}_0^+$	E2E bandwidth and delay requirement on path p
$\mu : \mathrm{V}(SG) \mapsto \mathrm{V}(RG)$ $\lambda : \mathrm{E}(SG) \mapsto \mathrm{P}(RG)$	Node and link mapping functions

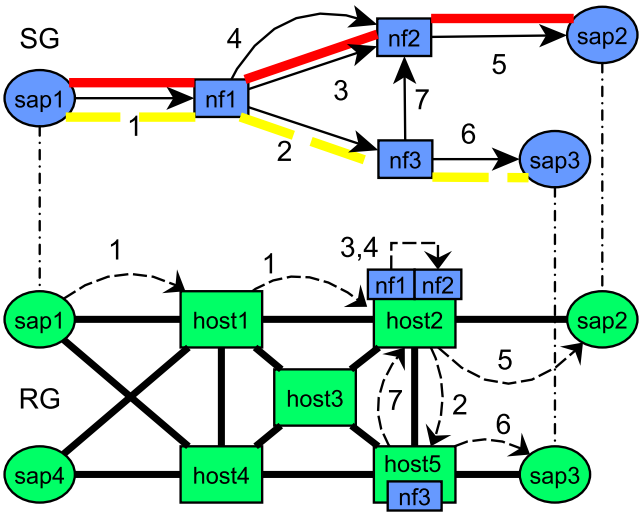


Figure 4: Illustration of Service Graph Embedding

A service request is given as a Service Graph (SG) and the computing and networking resources are described by the Resource Graph (RG). Users or other network domains are connected by Service Access Points (SAP) in both graphs. The notations used in the subsequent sections are summarized in Tab. 1, and an illustration for the SGE problem is given in Figure 4. A mapping of an SG to the RG solving an SGE consists of μ and λ functions which assign SG nodes and links to resources, respectively. Service Chains (SC) are defined as end-to-end (E2E) paths in the SG, where endpoints are represented by SAPs. The traversed SG nodes are Network Functions (NF), defining the semantics of the chain, and the corresponding bandwidth and delay requirements specify the QoS parameters for the Service Chain (see Tab. 1). The goal of the SGE is to minimize network resource utilization and maximize the number of embedded SGs. As a generalization of VNE, SGE is also $\mathcal{NP}$-hard [FSF; CRB09].

Some existing VNE algorithms can be used to partially solve an SGE problem, but generally, there are some aspects of SGE which are not supported by any of them. For instance, logical node collocation (mapping two neighbouring NFs to the same substrate node, e.g. $nf1$ and $nf2$ in Figure 4) is neglected by ViNEYard [CRB], although some approaches have been made to fix this [FSF], direct collocation support is desirable. Contrary to VNE, SGE considers the functional types of logical nodes and maps them according to substrate network capabilities. More importantly, the maximal allowed latency requirement on an E2E path is not supported by any VNE algorithm so far.

2.2.2 Algorithm

In this section, our algorithm is presented in hierarchical manner, focusing on the main ideas of the design.

Core processing. The orchestration algorithm greedily maps a Service Graph node and an adjacent link in one step (this pair is referred to as leg from now on, e.g. SG link 1 and $nf1$ in Figure 4). If such a pair cannot be mapped, the procedure backtracks for other possible mappings. The pseudo-code is shown in Alg. 1. An ordering on the legs is calculated by DivideIntoSubchains().

A leg is denoted by (e_{nf}, nf), where $e_{nf} \in \mathrm{E}(SG)$ terminates in $nf \in \mathrm{V}(SG)$, which are always mapped together, and this mapping is considered one greedy step (or backtracking step, in case of mapping failure) of the orchestration procedure. The mapping process iterates on each (e_{nf}, nf) legs and tries to map them based on customizable preference parameters using heuristics implemented by MapOneNF() in line 7.

Algorithm 1: SGE Algorithm

```
1  Procedure MAP(SG, RG, Ψ(SG)) ↪ μ, λ
2      SG, RG, G^RG_Ψ(SG) ← Preprocess()[SG, RG, Ψ(SG)]
3      Ψ_div(SG) ←DivideIntoSubchains()[SG, Ψ(SG)]
4      forall q ∈ Ψ_div(SG) do
5          forall (e_nf, nf) ∈ q do
6              while μ(e_nf) = ∅ or λ(nf) = ∅ do
7                  success ←MapOneNF()[e_nf, nf, q]
8                  if ¬success then
9                      (e_nf', nf') ←PreviousLeg()
10                     Undo mapping of (e_nf', nf')
11                     (e_nf, nf) ← (e_nf', nf')
12                 end
13             end
14         end
15     end
16     return μ, λ
```

If the embedding step fails, PreviousLeg() is used to retrieve the previous SG link - SG node pair, both denoted by primed variables in line 9. The existing mapping and resource reservation for (e'_{nf}, nf') is undone.

In case of successful mapping step, the resources are reserved for (e_{nf}, nf), the available delay of all affected Service Chains are updated and the mapping proceeds until all elements of SG are embedded to the Resource Graph and the result is returned.

Preprocessing. The Preprocess() of Alg. 1 prepares the input structures for the embedding process, and a helper structure is created. The end-to-end bandwidth requirement of Service Chains are added to link-wise bandwidth requirement of the SG, so from now on B_p parameter of all $c_p \in \Psi(SG)$ are incorporated by the link-wise bandwidth requirement of SG links on path $p \subseteq \mathrm{P}(SG)$. The lower architecture layer reports the total networking and computing resources, in addition to the currently running NFs on the infrastructure. So during the preprocessing of RG, the available resources are calculated respecting the reserved resources for previously deployed services.

The mapping of SAPs from the SG can be done unambiguously to the SAPs of RG, so this is calculated in Preprocess() method, stored in μ structure and kept fixed during the whole orchestration. The definition of the helper structure, calculated in the preprocessing stage is denoted by $\mathcal{G}^{RG}_{\Psi(SG)}$ and calculated as follows:

$$\mathcal{G}^{RG}_{\Psi(SG)} = \{G_{c_p} | G_{c_p} \subseteq RG, \forall c_p = (B_p, D_p, p) \in \Psi(SG), \\ \forall v \in \mathrm{V}(G_p) : d(p.first, v) + d(v, p.last) \leq D_p\}$$

It contains an induced subgraph of RG for each Service Chain, which gathers all the nodes of RG whose sum of distances from the endpoints[2] of the Service Chain measured in delay are not larger than the end-to-end delay requirement of the Service Chain. The matrix of d is calculated by Floyd-Warshall algorithm using edge delay as the weight function for the edges and nodes of RG, which needs to be done only once for a substrate network and d is cached for further orchestrations. In other words, the set $\mathcal{G}^{RG}_{\Psi(SG)}$ contains a subgraph of RG for every Service Chain separately, which should be used to host all NFs of the corresponding SC.

[2] $p.first$ and $p.last$ are always common SAP nodes of RG and SG.

Service Chain division. The goal of another important subroutine is to partition[3] the $\mathrm{E}(SG)$ into simple paths, called subchains, and to determine an ordering between these subchains, which helps the algorithm to map the (e_{nf}, nf) legs in predictable order. This is implemented by DivideIntoSubchains() function of Alg. 1.

The subchain division subroutine takes into account the delay requirements of the Service Chains, contained in $\Psi(SG)$, and ensures that the strict end-to-end chains are cut into less pieces, and mapped earlier during the mapping (so less placing possibility is compensated by less loaded RG).

An NF node of the Service Graph can be used by multiple Service Chains, so that for these NFs host restrictions are saved to respect the sharing for all using SCs. A shared NF can only be hosted on a Resource Graph node, which is contained by all subgraphs $G_{c_p} \in \mathcal{G}^{RG}_{\Psi(SG)}$ for all c_p Service Chains which uses this NF.

The subroutine's return value is $\Psi_{div}(SG)$, which is an ordered container for the disjoint paths of the calculated subchains.

One greedy step. The MapOneVNF() function's task is to map the given SG link – SG node pair to the best hosting RG path and node, in addition to saving some other good host candidates of the current (e_{nf}, nf) leg for backtracking. Its return value reports whether the greedy mapping step was successful or not.

Possible hosts for an NF must have enough resources, must be able to run the given NF type, must not be too far (measured in delay) from the termination of the end-to-end chain and must be accessible on a path which has enough bandwidth and must not be too far (measured in delay) from the previously used host.

An objective function value is calculated for every possible host $v \in \mathrm{V}(RG)$ and substrate path $p \in \mathrm{P}(SG)$, leading from the previously mapped leg's hosting node to v (i.e. $v = p.last$).

The objective function value of a potential $(e_{nf}, nf) \leftrightarrow (p, v)$ mapping is influenced by three components: $i)$ average link bandwidth utilization on path p; $ii)$ delay used by path p divided by the remaining delay budget; $iii)$ weighted sum of preference values of utilization for every resource type on host v.

Preference value of utilization on a host for a resource type can be defined by any real function $\phi_r : [0, 1] \mapsto [0, 1]$, which should quantify how much a utilization state is preferred. The bigger the preference value, the less the state is preferred. By default this function is 0 if utilization is less than 0.2, and linear between $\phi_r(0.2) = 0$ and $\phi_r(1.0) = 1.0$ for each node resource type.

The best host and path pair (p, v), i.e., the one with the lowest objective function value, is chosen for mapping, and a few other good candidates are saved for backtracking, in case of subsequent mapping failure.

2.2.3 Evaluation

The orchestration algorithm has been evaluated on a real world topology taken from SNDlib[4] [Orl+07], which has 42 nodes and 157 edges, representing access, aggregation and core network parts, equipped with computation resources. The backbone topology of Germany was chosen, which can be found with the name of dfn-gwin in the SNDlib database. The topology itself is shown in Figure 5, where the nodes relative position corresponds to their geographical placement.

Before using this topology in the testing, we have added some additional capabilities to the backbone network to represent the access network and data center parts in the substrate network. The access network consist of an access switch and some SAPs connected to them, which could represent the connection to smaller networks of the country-side, internal company networks or neighboring countries. The data center parts are host nodes directly connected to

[3] A partition of a set U divides U into disjoint subsets, whose union is U.

[4] The dfn-gwin topology was used with additional network parts (6 nodes have been attached to an access network) and computing resources (another 6 nodes have been attached to a data center).

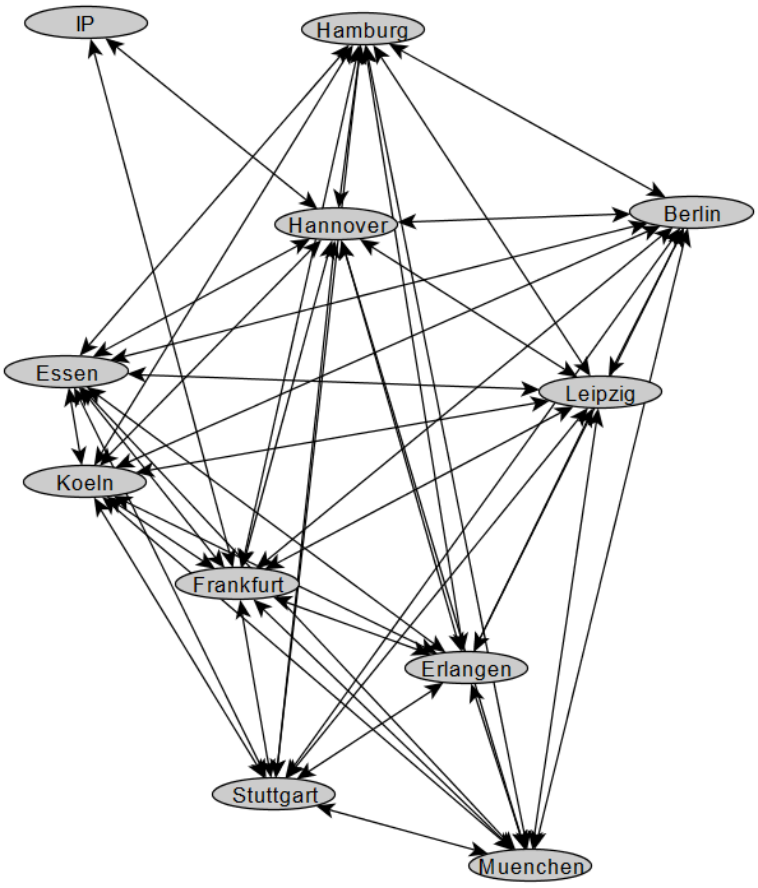


Figure 5: Backbone network of Germany (dfn-gwin) taken from SNDlib.

the backbone nodes, which represent the network providers internal cloud infrastructure or rented computation capacity from public clouds. These nodes provide the network function running capabilities of the network. The expanded topology was generated randomly with some given starting constraints, the result is shown in Figure 6. The backbone network topology is the same as Figure 5, but this time they are not depicted according to their geographical location.

The nodes' naming convention in Figure 6 is straightforward, it represents the node types. The backbone nodes and the access switches are shown in blue rectangles, the hosts are green parallelograms and the SAPs are red ovals. Both the switches and hosts have $40,000$ Mbps internal switching capacity. Both the switches and hosts have delay contribution with 1.0 ms and 0.5 ms respectively for forwarding the traffic between their arbitrary two ports. Furthermore, each host has $320,000$ MB memory 400 CPU and $1,500$ GB storage, which can be seen as the aggregate capacity of a private cloud network. Each host supports 6 types of network function, which are chosen randomly from a given set of 10 different network function types. The links are divided into three groups: (1) core links between the backbone nodes, (2) aggregation links between an access switch and a backbone node, (3) access links between the SAPs and the access switches. Their attributes are shown in Table 2.

Table 2: Link attributes for each link type in the expanded SNDlib topology.

Link type	Bandwidth (Mbps)	Latency (ms)
Core link	$10,000$	1.0
Aggregation link	$1,000$	0.5
Access link	100	1.0

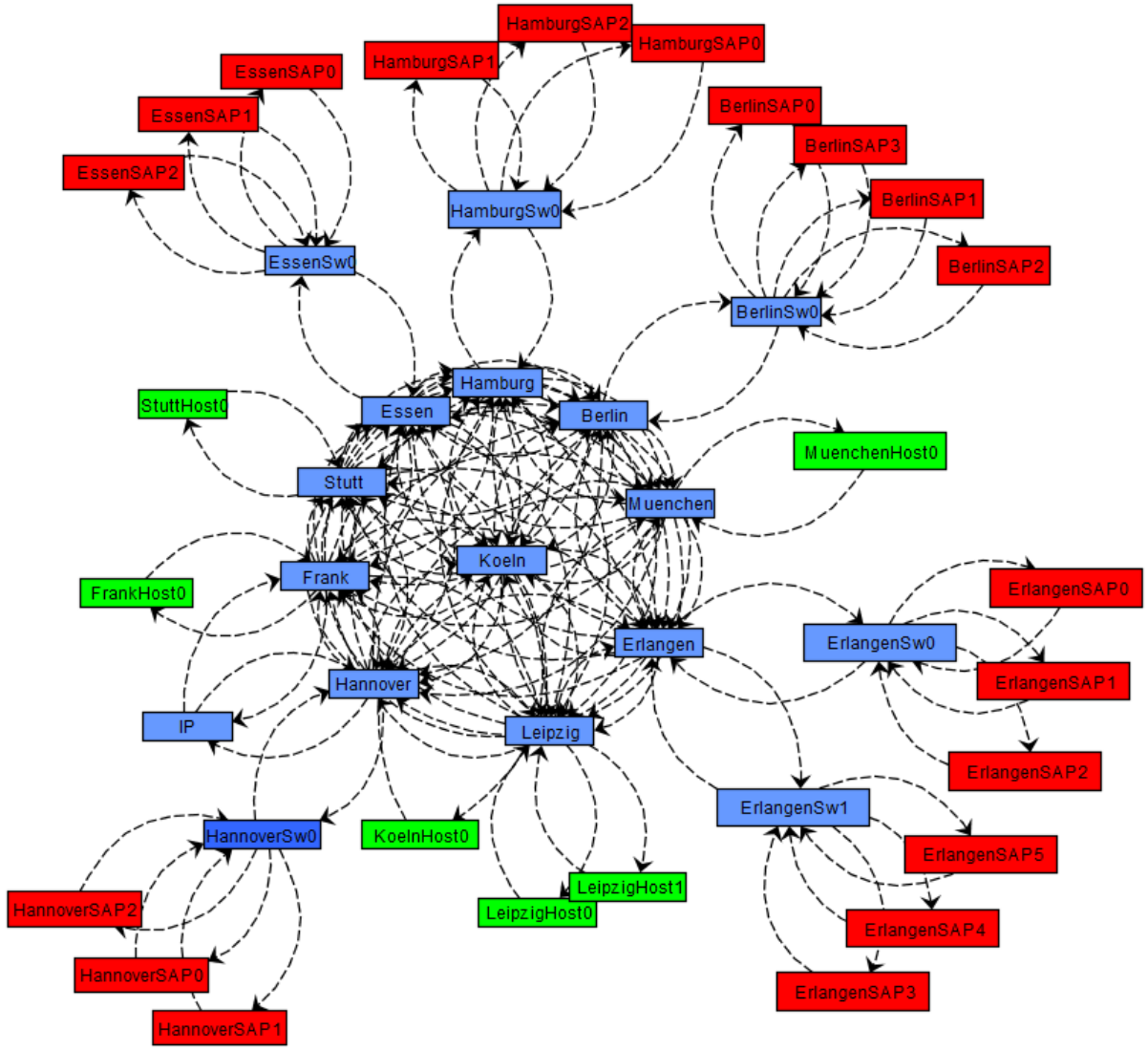


Figure 6: Expanded dfn-gwin topology representing access and data center network parts in addition to the backbone network.

A benchmark test has been conducted on the orchestration algorithm, by defining a sequence of service graphs, which shall be mapped onto a shared substrate network. The algorithm should continue the mapping until it refuses to map a request. At this state the load state of the network can be investigated and arbitrary metric or graph can be generated, furthermore, the number of successfully mapped service graphs characterizes the mapping performance. The sequences can be randomized for each execution, but based on a given seed for the random generator, the exact same service graph sequence with identical resource requirement for each node and resource type can be reproduced. The main difference from the acceptance ratio is the customizability of the service graph sequence. Many attributes of the service graph sequence can be set, such as size and structure of the service graphs, the resource requirements of the network functions and the strictness of end-to-end latency requirements. The service graph benchmark sequence can be used to compare the mapping results of different algorithms in scenarios with predefined attributes. A given state of the substrate network can be investigated in detail by the Cumulative Distribution Functions (CDF) of the resource utilization of network elements (links, host, forwarding nodes) for each of the resource types. The CDF can be used to plot how many of the network elements (vertical axis) have a resource utilization lower than a given value in percentage

Table 3: Minimum and maximum values of the uniform distributions for resource requirement generation of the service graph sequences

Resource type	Minimum value	Maximum value
Number of CPUs	1	4
Memory (MB)	0.0	1600.0
Storage (GB)	0.0	3.0
End-to-end latency (ms)	60.0	220.0
End-to-end bandwidth (Mbps)	0.0	7.0

(horizontal axis). Examples are shown in this section.

Simple Service Chains (SC) with 1 to 8 NFs connected between two SAPs have been generated uniformly with randomized resource requirements with the same distribution. The parameters of the uniform distributions are shown in Table 3. The resource requirement were relatively small compared to the available resources, and approximately 1000 NFs could be hosted on the whole network, limited by the most scarce resource type. This can be used to compare the performance to other algorithms or the different settings of the presented algorithm. The generated SC sequences have been cumulatively mapped to the network in batches of four (in other words, one Service Graph consists of four SCs), so the orchestration algorithm had to provision the request with monotonically decreasing available resources.

Internal latency and bandwidth requirement of the network functions are omitted. Also the link bandwidth and latency requirement are left unset, meaning there is no requirement for them. The resource requirements are relatively small compared to the available resources. In the current setting the scarcest resource is the number of CPUs. In total, there are $6 * 400 = 2,400$ CPU in the whole substrate network, while one service chain's expected total requested CPU count can be calculated from the expected values of service chain length and requested CPU count by one network function: $4.5 * 2.5 = 11.25$. From this we can calculate how many service chains can be mapped at most onto the shared resource graph: $2,400/11.25 = 213.33$. In the simplest case the service chains of the previously described benchmarking sequence arrived one-by-one for the algorithm. The service chains do not timeout from the substrate network, in other words they have infinite lifetime, so the load gets bigger monotonically.

Figure 7 depicts the conformation of CPU and memory resources throughout the whole test sequence of SG re-

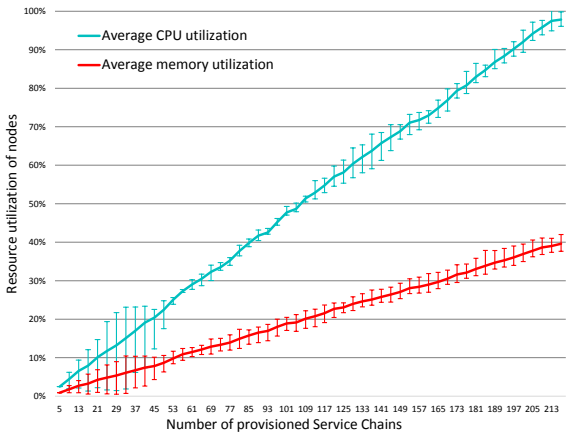
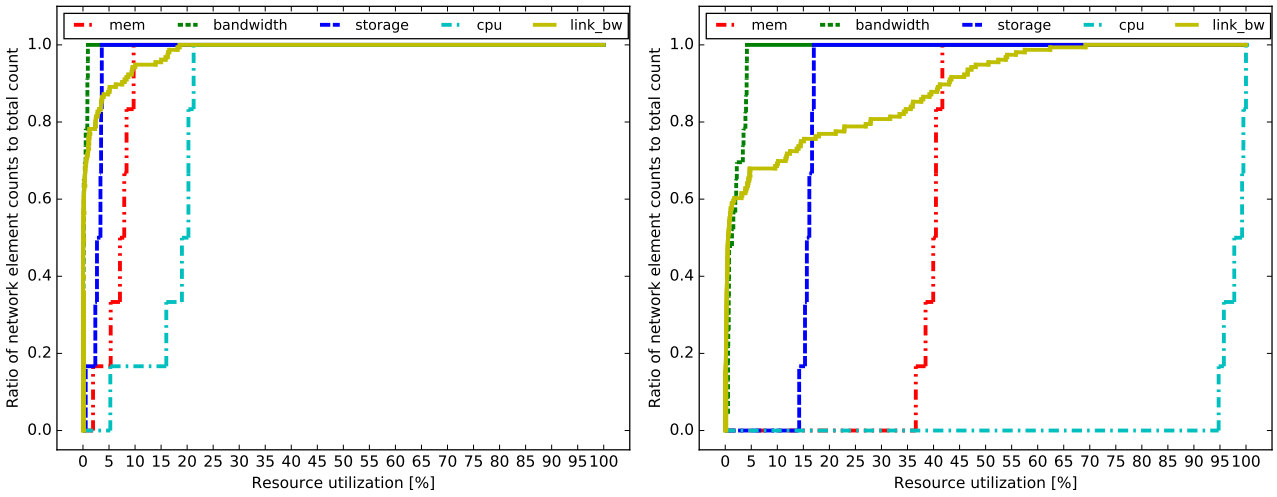


Figure 7: Minimal, maximal and average utilization of CPU and memory under increasing load.



(a) Less loaded state.

(b) Almost fully loaded state in term of CPU resource.

Figure 8: Cumulative Distribution Function of various resource utilizations .

quests (on the extended dfn-gwin topology). The horizontal axis shows the number of mapped SCs, and the vertical axis shows the minimal, average and maximal resource utilization of the resources. According to the expectations, the resource utilizations increase monotonically, until the utilization of CPU reaches its maximal capacity. As we show in 7, somewhere above 200 mapped service chains the average CPU utilization converges towards 100%, meaning that every host node with CPU capacity is close to full load. The algorithm cannot map much more than 200 service chains, as it was expected based on the previously explained preliminary calculations. The memory utilization of the network is nowhere near to its fully loaded state, but later arriving service chains can request any kind of resource types, so the mapping fails soon after due to the fully loaded CPU resource state.

Figure 9 shows the load state of Resource Graph at two different moments of the benchmarking test. The Cumulative Distribution Function (CDF) of all resource types are plotted in both cases. A CDF of a given resource type shows the ratio of network elements which have their resource utilization lower or equal to the value on the horizontal axis. Resources of the computation nodes are CPU, memory, storage and bandwidth, while forwarding nodes have only bandwidth resource (i.e., the CDF of bandwidth has much more steps). The plotted link resource is bandwidth, denoted by link_bw in Figure 9.

Node resource CDFs show that our orchestration algorithm balances the load nicely throughout the network. For instance, the CDF of memory in Figure 8b says all nodes are utilized between 35 and 45 percent. The CDF of CPU shows that it is the most scarce resource in the network, because it is the rightmost curve in both figures.

Figure 8a shows a less loaded state during the same service graph benchmarking test, when only 37 service chains are provisioned. The Cumulative Distribution Functions of each resource type are depicted with different colors and patterns. For instance, the CDF of the CPU (denoted by teal squares), shows that the 1/6th of the hosts are below 10% of CPU utilization, while all of the hosts are below 20%. Figure 8a also shows that the algorithm keeps the load distributed in the network to maximize the acceptance value of the next service graph (which in this case contains only one service chain).

The CDFs of the different resource types have different step sizes. This is caused by the difference in the numbers of network elements which own a resource type. For example, there are only 6 nodes which have CPU, memory and storage, while there are more nodes with internal bandwidth, and finally there are more than 100 links in the network. The next CDF collection in Figure 8b shows a fully loaded state after more than 200 service chains were mapped. This

Table 4: Performance comparison of the two mapping approaches based on 100 different service graph sequences.

	Average result	15th %ile	85th %ile
MILP-based algorithm	207.37	200	216
Heuristic algorithm	131.57	119	163

graph is also generated from the output of the same service graph sequence, which provided the data for the previous two pictures. Figure 8b shows that it is not really possible to map more service chains onto this substrate network. In terms of CPU resource, this network is fully loaded, because every node's utilization is between 95% and 100% percent. Furthermore, this figure shows that other resource components are kept relatively well balanced.

The tests were executed on general personal computers, and the runtime of a whole test sequence (which consists of approximately 900 NF mappings onto the extended SNDlib topology) is around 38 seconds, including all the preprocessing and the output generation, and excluding the Floyd-Warshall algorithm. **Our orchestration algorithm's runtime scales polynomially with the input size.**

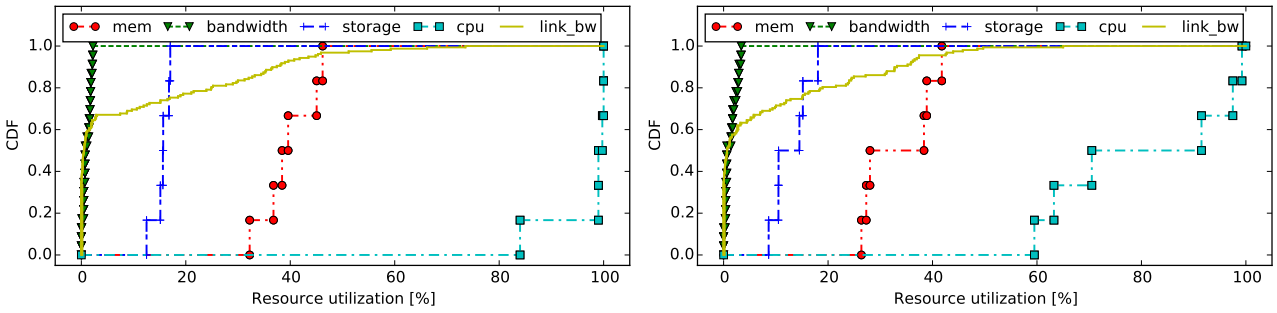
As our algorithm is based on heuristics and provides a suboptimal solution it is important to analyze how close are the results of the mapping process to other algorithms or the optimal solution. For this reason, we have implemented a Mixed-Integer Linear Programming (MILP) based orchestration algorithm.

The MILP-based algorithm uses the formal problem definition of service graph embedding, presented earlier. The implementation uses the Python API of the Gurobi optimization library. The objective function of the linear programming is maximizing the number of embedded service graphs and minimizing the link resources used. First of all, let us compare the MILP-based and heuristic based algorithm by the primary metric of the service chain benchmarks, namely the number of successfully mapped service chains. The parameter settings of the test sequence are the same as presented in Table 2 and Table 3, but this time multiple sequence instances are generated by different seeds given to the random generator module. The sequences are mainly the same, all of their main features are identical, but the actual values for the requirements are different.

The sequences were mapped in batches of four by the heuristic algorithm, while the MILP-based solution mapped the whole test sequence at once. In the latter case if the algorithm could successfully map the first N service chains, than it started to map the exact same service chain sequence from the beginning with a completely unloaded resource graph. This way we can compare the performance of the online and offline approaches. This is practical, because the heuristic algorithm cannot really profit from an extremely long service chain batch (as far as its greedy mapping procedure cannot backtrack until the very beginning of the whole batch), while the MILP-based solution was designed for such offline operation. 100 different seeds were used to compare the mapping performance of the two approaches. The results are shown in Table 4.

The calculation of the percentile values are done as follows: the 100 results are ordered in ascending order, the (15% of 100) 15th value is the 15th percentile, while the 85th result in the order is the 85th percentile. This tells us that the heuristic algorithm with the online approach and low computation needs can generally achieve 2/3 of the performance of the offline orchestration procedure. Furthermore, the percentiles show us that the distribution of the results are not dispersed too much (about 10%-30% from the average result), which is a good attribute of the mappings.

The quality of the mappings can be compared by the resource utilization CDFs of the substrate network states after the orchestration algorithms failed to successfully map a service chain batch. The two CDF collections can be seen in Figure 9a and Figure 9b, showing the resulting network states of the MILP-based and the heuristic based algorithms. Figure 9a reflects well the link bandwidth optimization goal of the MILP-based solution, because its CDF of link resource is a bit more steep than the other one. Furthermore, the two figures show that the heuristic approach does not fall much behind in the load balancing capability, especially in the case of less scarce resources such as node bandwidth, memory



(a) The CDF of the network resources after the MILP-based algorithm failed to map a service chain after 212 successfully provisioned service chains (the chains were arriving batched together from the beginning).

(b) The CDF of the network resources after the heuristic algorithm failed to map a service chain after 180 successfully provisioned service chains (the chains were coming in batches of four).

Figure 9: CDF of resource utilizations after mapping as many SCs as the orchestration algorithms could.

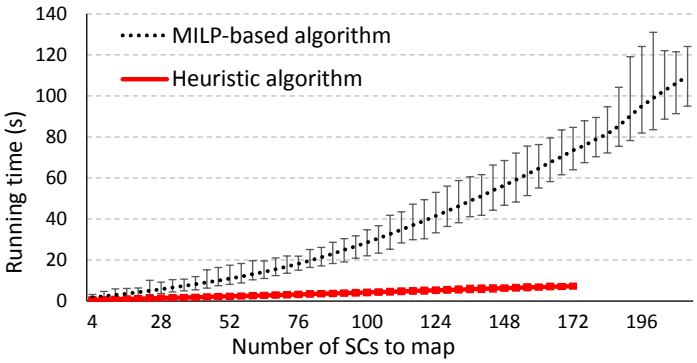


Figure 10: Runtime comparison of the two mapping approaches based on increasing input service graph size.

or storage. In Figure 9b, there is relatively much area below the CDF of CPU, which means there is still a significant amount of CPU resource left in the network, but the algorithm failed to map at this state with this example service chain sequence. It could indicate that the algorithm has not searched enough possible mappings. Setting a higher preference value for the CPU resource, or setting a bigger depth limit for the backtracking could help in making the mapping result better.

Finally, let us compare the runtimes of the two approaches. To make this comparison fair, both algorithms map the whole request containing several service chains in one step. This ensures that the two algorithms receive the same size of input service graphs (and also all other parameters are completely identical). Similarly to the previous tests, the substrate graph is shared between all the service chains and the topology is kept fixed during the test sequence. The average, minimum and maximum runtime of the 100 service chain sequences are delineated in Figure 10. The simulation was conducted on computers with Intel Core i5 processors and 8GB of RAM. As we expected, the heuristic algorithm exhibits the polynomial scaling with the number of input service chains, while the MILP-based approach imposes impractical runtimes and exhibits worse scaling behavior.

Our algorithm has been integrated with ESCAPE and it was used in several multi-domain experiments. The evaluation and the conducted experiments confirmed that the proposed algorithm provides a fast, efficient and scalable solution for the problem of automatic resource optimization in a virtualization-based and software-controlled network architecture. In addition, the presented evaluation results demonstrate the load balancing capability of our solution. Altogether, our Orchestrator logic is ready to be tested and deployed in real-life scenarios.

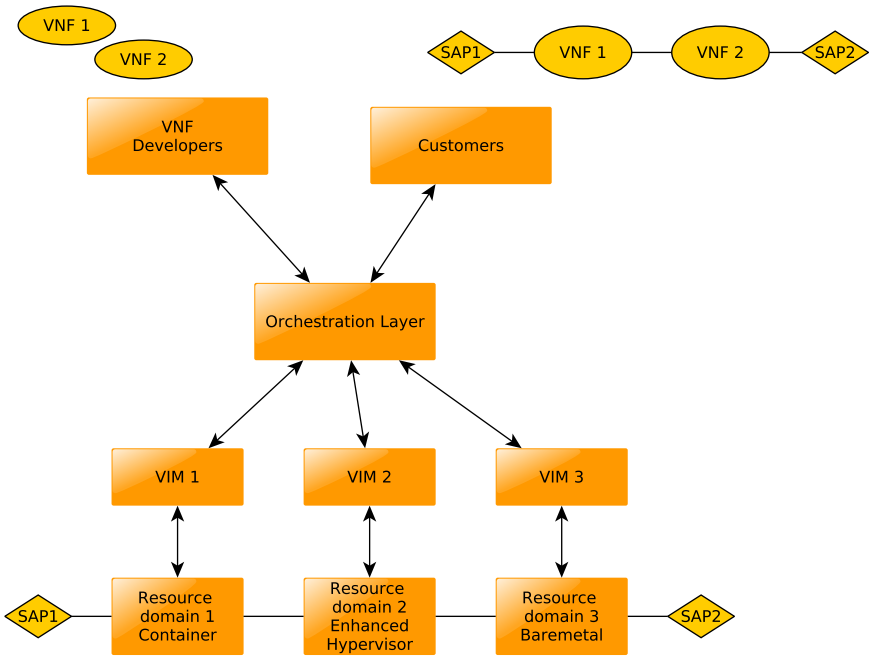


Figure 11: NFV MANO: Customers and VNF Developers

2.3 VNF benchmarking

Network Function Virtualization (NFV) pursues delivering Virtualized Network Functions (VNFs) in NFV Infrastructure (NFVI) where requirements of portability, performance, elasticity, among others, are demanded by carriers and network operators [ETS14a]. Particularly, in the short term, performance and elasticity are important factors to realize NFV concepts [Han+15]. Associated to these goals VNFs need to be deployed with predictable performance, taking into account the variable processing overheads of virtualization and the underlying available NFVI resources.

In essence, NFV Management and Orchestration (MANO) plane manages infrastructure and network services resources [ETS14c]. This involves taking into account performance considerations for NF-FGs allocation (e.g., fulfilling NFVI resources by orchestration demands). Furthermore, when attempting to manage service assurance of embedded NF-FGs [Pai+15] MANO tasks may depend on analytics measurements for scaling and migration purposes. As a consequence, NFV orchestration managers (e.g., NFVO, RO) need a coherent understanding of performance vs. resource needs for VNFs specific to Infrastructure/Resource domains (e.g., NFVI Points of Presence - PoPs).

Let us take an example where a VNF Developer creates a new code base for a VNF. She is able to optimize her VNF for multiple execution environments and offers these as a set of different but equivalent implementations of VNF1 for Service Providers (SPs) (see Fig. 11). Orchestration managers (e.g., Network Service Orchestrator) must know adequate infrastructure resources to allocate when deploying VNF1, specially if it belongs to a network service (NF-FG) with a target SLA, e.g., min throughput or max latency. However, instances of VNF1 optimized for heterogeneous Infrastructure/Resource domains may need different resources/settings for the same behavior (performance). Orchestration managers (e.g., Network Service Orchestrator, RO) need to take into account all these requirements to embed VNFs and allocate proper infrastructure resources in accordance with network service intents (SLAs).

In what follows we define the problem and challenges, the related terms and introduce a framework proposal.

2.3.1 Problem Statement and Challenges

Previous section motivates orchestration managers (e.g., Network Service Orchestrator, RO) needs of knowledge about correlations between VNFs performance and Infrastructure/Resource domain resources.

Suppose that specifications, based on developers' definitions, exist in ETSI's VNF Descriptors [ETS14b] associating performance to specific Infrastructure/Resource domain resources. Such VNF, when deployed, may present oscillations in its performance because of changes in the underlying infrastructure (due to varying customers workloads, traffic engineering, scaling/migration, etc). Therefore, we believe we need descriptions on how to benchmark VNFs' resources and performance characteristics (e.g., VNF Profiles) on demand and in a flexible process supporting different Infrastructure/Resource domains, which could create service level specification associations.

Joint definitions of multiple VNFs Descriptors in different environments can compose a unique information base (e.g., NF-IB) for capabilities descriptions of supported VNFs on the evaluated environments as well their corresponding expected performance. Then, orchestration managers (e.g., Network Service Orchestrator, RO) can consult such information base to retrieve profiles of VNFs and use them to optimize scheduling/embedding algorithms.

Some of the associated challenges are:

- Consistency: Will the VNF deployed at a Infrastructure/Resource domain deliver the performance given in the VNF Profile for the Infrastructure/Resource domain under different workloads? The performance of a VNF may not only depend on the resources allocated to the VNF but on the overall workload at the target Infrastructure/Resource domain.
- Stability: Benchmark measurements need to present consistent results when applied over different Infrastructure/Resource domains. How express benchmarking descriptions to fit in different virtualized environments? How to uniformly handle service/resource definitions and metrics?
- Predictability/Accuracy: A given VIM / Infrastructure/Resource domain may virtualize (i.e., hide) certain resource details; hence a benchmark evaluation might be executed in a different resource set of the Infrastructure/Resource domain unlike the final production environment. How well benchmark results on controlled scenarios can correspond to and represent actual workloads in real virtualized environments?
- VNF benchmarking service description: How to define the VNF benchmarking process at the abstraction of the corresponding component (e.g., VIM)? How to offer benchmarking as a service to be consumed by the different stakeholders?
- VNF benchmarking versus monitoring: How can a VNF benchmarking process complement continuous monitoring of NFVI? When does a VNF benchmarking process surpass in performance and costs the execution of continuous monitoring process? And for which type of Infrastructure/Resource domains and/or VNFs is it necessary/sufficient?

2.3.2 Proposed Approach

Definition: VNF Benchmarking as a Service (VBaaS) – a standalone service that can be hosted by orchestration managers (e.g., Network Service Orchestrator, RO) to report a VNF Profile. VBaas might take inputs consisting of NF-FGs determining VNFs to be evaluated under specific NFVI targets. As output, VBaaS returns a Service Graph based on the VNF-Benchmark Profile (VNF-BP) containing different parameters used to evaluate all required candidates (Infrastructure/Resource domains) for the specified VNF. Such interactions are abstracted by the VBaaS API. A VBaaS Service

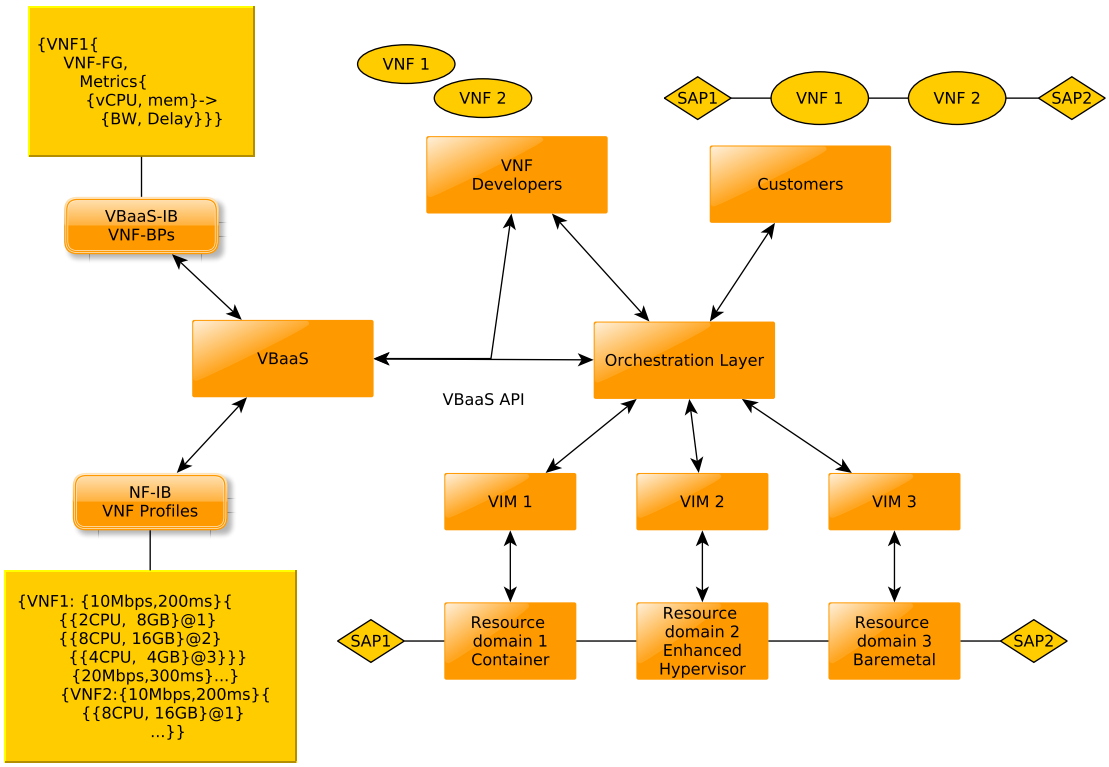


Figure 12: NFV MANO and VBaaS Approach

Graph is deployed by orchestration managers (e.g., Network Service Orchestrator, RO), VBaaS receives benchmarking results and builds VNF Profiles. VNF performance values or VNF resource values can be provided, and VBaaS can only complement the NF-FG missing parts to build VNF- BPs, or report an existing up-to-date VNF Profile for pre-defined values.

In comparison with Fig. 11, the VBaaS approach introduces two information bases, a VBaaS component and an API to interact with orchestration managers (e.g., Network Service Orchestrator, RO), as shown in Fig. 12. On one hand, the Network Function Information Base (NF-IB) holds the VNF Profiles and on the other hand the VBaaS-IB holds the VNF Benchmark Profiles (VNF-BPs).

Definition: VNF Profile – is a mapping between virtualized resources of a certain Infrastructure/Resource domain (e.g., virtual machine with vCPU, memory) and VNF performance (e.g., throughput, delay, packet loss between in/out or ports).

An orchestration function can use the VNF Profile to select hosts for VNFs in a NF-FG and to allocate the necessary resources to deliver a given (predictable) service quality (SLA).

Fig. 12 shows an example VNF1 and VNF2, which are implemented by a VNF Developer, who provides metrics for VBaaS build VNF-BPs. The VNF-BP is then used by VBaaS when orchestration managers (e.g., Network Service Orchestrator, RO) needs to evaluate the available Infrastructure/Resource domains for different resources configuration. In this case, Network Service Orchestrator instantiates the NF-FG (VBaaS Service Graph) as defined in VNF1-BP; configures the variable parameters in the configuration template, e.g., CPU and memory values and handles the configuration file to the measurement controller (part of the NF-FG definition) to execute the benchmarking. The measurement controller make the measurement agents execute the benchmark tests (the execution environment might also be monitored by the agents) and reports the results back to the VBaaS. As a result, resource and performance associations for

the VNF is created for the evaluated the Infrastructure/Resource domains into the NF-IB. Fig. 12 shows, as an example, that VNF1 needs 2CPU (cores) and 8GByte of memory to handle up 10Mbps input rate with transaction delay less than 200ms at Infrastructure/Resource domain 1.

On the left side of the orchestration managers (e.g., Network Service Orchestrator, RO) in Fig. 12 is the resulting NF-IB by the application of VBaaS-IB profiles on the extraction of VNFs performance characteristics in different Infrastructure/Resource domains. Basically, a VNF profile corresponds to performance metrics in different environments. For example, VNF1 to offer 10Mbps bandwidth with 200ms latency needs (if deployed) in Infrastructure/Resource domain 1 2CPU (cores) and 8GB of memory, meanwhile in Infrastructure/Resource domain 2 requires 8CPU and 16GB. Similar results can be presented for VNF2, if catalogued as a VNF profile. In this scenario, VBaaS can be seen featuring VNFs in different Infrastructure/Resource domains in an on-demand composition of profiles, for example, to be used in provisioning NF-FGs containing VNF1 or VNF2, attending Infrastructure/Resource domains 1, 2 or 3 resources consumption. Such profiles present performance metrics of VNFs in different Infrastructure/Resource domains when, for example, orchestration managers (e.g., Network Service Orchestrator, RO) need to deploy particular NF-FGs that contain elements already certified in NF-IB profiles.

Definition: VNF Benchmarking Profile (VNF-BPs) – the specification of how to measure the VNF Profile for a given VNF. The specification includes structural (e.g., measurement components defined as a NF-FG) and functional (e.g., measurement process definitions and configurable parameters) instructions, and variable parameters (metrics) at different abstractions (e.g., vCPU, memory, bandwidth, delay; session, transaction, tenants, etc.).

Note: a VNF-BP may be specific to a VNF or applicable to several VNF types.

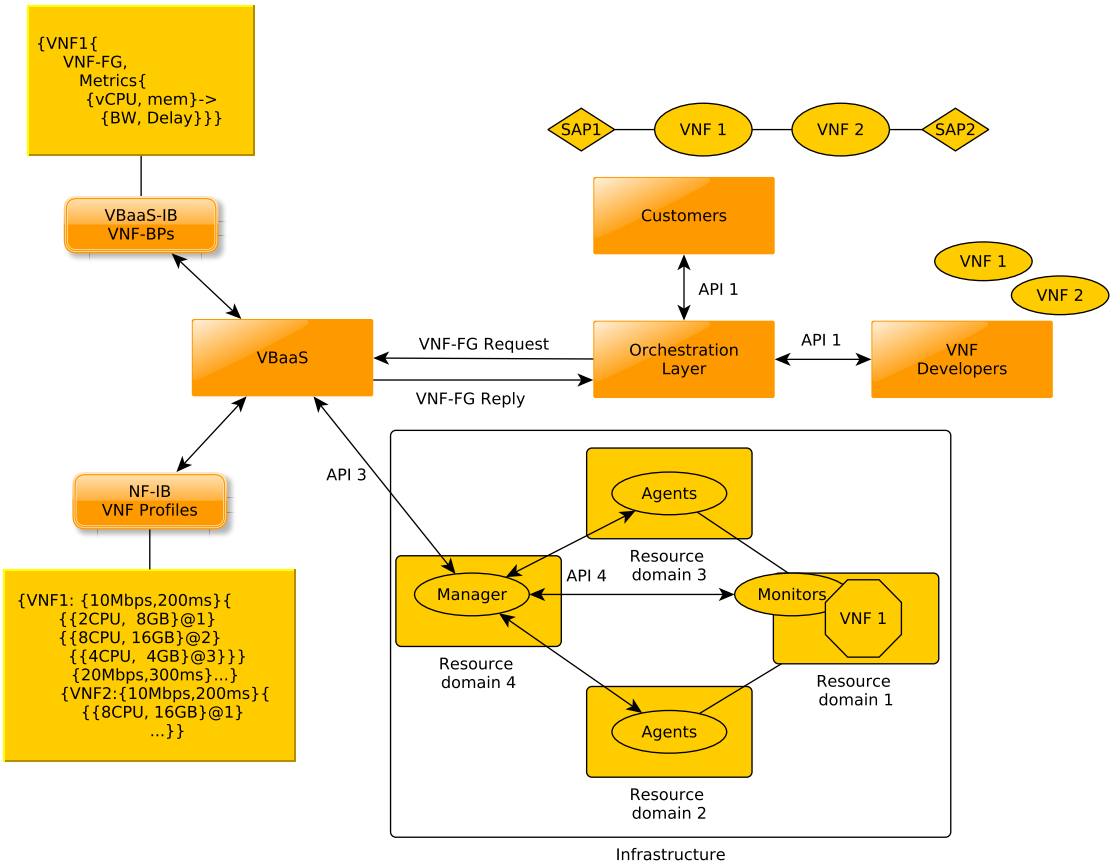


Figure 13: VBaaS APIs

Fig. 13 presents details of VBaaS API regarding the interactions with orchestration managers (e.g., Network Service Orchestrator, RO) and the construction of VBaaS Service Graphs based on VNF- BPs. As a NF-FG (deployment request), Network Service Orchestrator/RO demands VNF1 allocated to a specific Infrastructure/Resource domain 1 and the remaining virtualized view of the current resources for the NF-FG deployment (e.g., interconnected Infrastructure/Resource domains). VBaaS (Manager Service) looks for VNF1-BP in VBaaS-IB, complements the requested NF-FG with benchmark instances (described below) to perform evaluations and monitor resources in VNF1 and Infrastructure/Resource domain 1. Network Service Orchestrator/RO receiving a NF-FG reply (deployment order of a VBaaS Service Graph) instantiates it as a normal provisioning process of network services. Benchmark reports of the deployed NF-FG are sent to VBaaS and it defines the construction of the VNF-Profile associated with that evaluation.

We believe in the existence of VBaaS instances (VNFs) which compose a VBaaS Service Graph to perform probing and monitoring activities on selected targets (VNF in a specific Infrastructure/Resource domain). Those are defined as Manager, Monitor and Agent, described in details below.

- Agent – executes active probes using benchmark tools to collect network and system performance metrics by interacting with other agents. While a single Agent is capable of performing localized benchmarks (e.g., Stress tests on CPU, memory, I/O), the interaction among distributed agents enable the generation and collection of end-to-end metrics (e.g., packet loss, delay). Consider here the possibility of one end be the VNF itself where, for example, one-way packet delay is evaluated. For instance, it can be used for probe measurements of throughput and latency in a multi-point distributed topology. Agent's APIs are open and extensible (e.g., to plug-and-bench new tools and metrics) and receive configuration and run-time commands from the Manager for synchronization purposes (e.g., test duration/repetition) with other Agent and Monitor instances.

- Monitor – when possible it is placed inside the target VNF and Infrastructure/Resource domain to perform active and passive monitoring and metric collection based on benchmarks evaluated according to Agents' workloads. For instance, to monitor CPU utilization of Infrastructure/Resource domain considering packet length traffic variations sent by Agents. Different from the active approach of Agents that can be seen as generic benchmarking VNFs, monitors observe particular properties according to Infrastructure/Resource domains and VNFs capabilities. Monitors can plug- and-bench new tools/metrics and maintain an interface with Manager to synchronize its activities with Agents. Besides, they can be instantiated, if allowed, inside the target VNF (e.g., as a plug- in process in a virtualized environment) to check internal behaviours, alongside the VNF deployment environment, i.e., Infrastructure/Resource domain, or in both places.

- Manager – is responsible for (i) the coordination and synchronization of activities between agents and monitors, (ii) collecting all benchmark raw results, and (iii) aggregating the inputs to construct a profile report that correlates different metrics as required by the VNF-BP. Therefore, it executes the main configuration, operation and management actions to deliver the results as specified by Infrastructure/Resource domains or VNF-BPs (e.g., instantiation of agents and monitors activities along- side tools and metrics configuration). Manager uses APIs to read and set configuration parameters for benchmarking instances such as metric parameters (e.g., bandwidth or packet loss probes) according to the VBaaS process. APIs are also used to receive benchmark instructions from VBaaS and send the reports of finished benchmark procedures.

The benchmarking process, however, can be offered as a service (see API (1) in right side of Fig. 13). For example, orchestration managers (e.g., Network Service Orchestrator, RO) may be able to handle a VBaaS-IB and offer a VNF Benchmarking as a Service to its client, e.g., another Network Service Orchestrator/RO. Similarly, a second Network

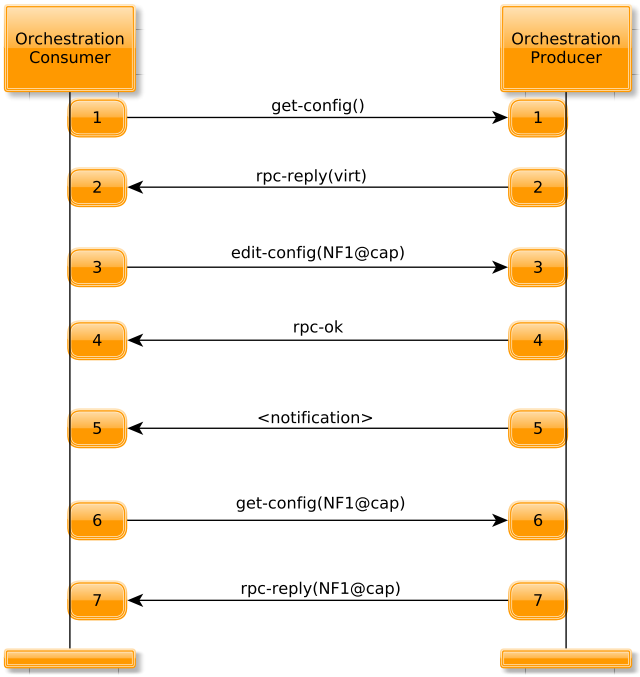


Figure 14: Sequence Diagram of VBaaS API (1) - Multi-Domain Interaction

Service Orchestrator/RO may store benchmark measurements in its NF-IB so it can report it to multiple clients if it shares resources among other orchestration managers. Alternatively, an Network Service Orchestrator/RO may offer to it's developers the VNF Benchmarking as a Service, to assess the performance of a VNF in the operator's environment. This may be part of a DevOps primitive (e.g., VNFs continuous integration).

2.3.3 Use Case: Unify Modelling

In UNIFY a generic description of compute and network is represented through the virtualizer model in YANG [SQK16]. It allows the aggregation of resources (e.g., nodes capabilities and NF instances) and their abstractions in a structure nicknamed Big Switch with Big Software (BiS-BiS) (joint software and network resource abstraction with a joint control API), which allows recursive control plane structuring for multi-domain hierarchies. For example, supported network functions of a joint compute and network virtualization (a BiS-BiS node) can be described with their associated resources and performance mapping, e.g., cpu, memory, storage -> bandwidth, delay (see Listing 2 based on virtualizer xml example in [SQK16]).

In this example, the interface between Network Service Orchestrators (Producer and Consumer of VBaaS API (1)), as shown in Fig. 13, is described by the UNIFY virtualization model [SQK16]. Fig. 14 shows an example netconf message exchange over the domain virtualization model. First the consumer reads in (1) and (2) the virtualization view, which includes the supported NFs reported in the capabilities section. Next in (3) the consumer requests a capability report of NF1 at a given virtualized node (UUID001), by defining the required delay and bandwidth performance parameters of the NF1 (see Listing 2). This is a dimensioning request according to [ETS16], i.e., the VBaaS service must evaluate and provide cpu, memory and storage values at the given virtualized node to deliver the given network performance characteristics. The start of the VBaaS process is acknowledged by a (4) rpc-ok. Once the VBaaS process completes a notification is sent (5) to the consumer, which reads in (6) and (7) the updates to the NF1 capability report, i.e., the missing cpu, memory and storage parameters (see Listing 3).

Following the above example and the terminology of [ETS16], the different tasks of performance verification, benchmarking and dimension can be mapped as follows:

- Verification: Both cpu, memory, storage and delay, bandwidth parameters are provided and the VBaaS system verifies if the given association is correct or not. In the case of a failure the entry might be removed or updated with correct values.
- Benchmarking: Where cpu, memory, storage parameters are provided and the VBaaS service fills-in the corresponding delay, bandwidth performance parameters. Note, such request might create more entries, like an entry with minimal delay and an entry with maximum bandwidth.
- Dimensioning: Where delay, bandwidth performance parameters are provided and the VBaaS service fills-in the corresponding cpu, memory, storage resource parameters (as shown in the above example).

Listing 2: VBaaS-Request(NF-FG-1): xml view

```
< virtualizer xmlns="http://fp7-unify.eu/framework/virtualizer">
  <id>UUID001</id>
  <name>Single node simple VBaaS request</name>
  <nodes>
    <node>
      <id>UUID11</id>
      <name>single Bis-Bis node</name>
      <type>BisBis</type>
      <ports>
        ...
      </ports>
      <resources>
        <cpu>12</cpu>
        <mem>64 GB</mem>
        <storage>20 TB</storage>
      </resources>
      < capabilities >
        <supported_NFs>
          <node>
            <id>NF1</id>
            <name>vFirewall</name>
            <type>Stateful virtual firewall C</type>
            <ports>
              <port>
                <id>1</id>
                <name>in</name>
                <port_type>port-abstract</port_type>
                < capability >...</ capability >
              </port>
              <port>
                <id>2</id>
                <name>out</name>
                <port_type>port-abstract</port_type>
                < capability >...</ capability >
              </port>
            </ports>
            <resources>
              <cpu> ? </cpu>
              <mem> ? </mem>
              <storage> ? </storage>
            </resources>
```

```
            <links>
              <link>
                <id>int0</id>
                <name>internal  horizontal</name>
                <src>../../ports/port[id=1]</src>
                <dst>../../ports/port[id=2]</dst>
                <resources>
                  <delay>20 us</delay>
                  <bandwidth>10 GB</bandwidth>
                </resources>
              </link>
            </links>
          </node>
        </supported_NFs>
      </capabilities>
    </node>
  </nodes>
</virtualizer>
```

Listing 3: VBaaS-Report(NF-FG-2): xml view

```
<virtualizer xmlns="http://fp7-unify.eu/framework/virtualizer">
  <id>UUID001</id>
  <name>Single node simple VBaaS report</name>
  <nodes>
    <node>
      <id>UUID11</id>
      <name>single Bis-Bis node</name>
      <type>BisBis</type>
      <capabilities>
        <supported_NFs>
          <node>
            <id>NF1</id>
            <name>vFirewall</name>
            <type>Stateful virtual firewall C</type>
            <ports>
              <port>
                <id>1</id>
                <name>in</name>
                <port_type>port-abstract</port_type>
                <capability>...</capability>
              </port>
              <port>
                <id>2</id>
                <name>out</name>
                <port_type>port-abstract</port_type>
                <capability>...</capability>
              </port>
            </ports>
            <resources>
              <cpu>2</cpu>
              <mem>8 GB</mem>
              <storage>20 GB</storage>
            </resources>
            <links>
              <link>
                <id>int0</id>
                <name>internal  horizontal</name>
                <src>../../ports/port[id=1]</src>
                <dst>../../ports/port[id=2]</dst>
```

```
                    <resources>
                      <delay> 20 us </delay>
                      <bandwidth> 10 GB </bandwidth>
                    </resources>
                  </link>
                </links>
              </node>
            </supported_NFs>
          </capabilities>
        </node>
      </nodes>
    </virtualizer>
```

2.3.4 Related drafts and open source projects

VBaaS intersects IETF Benchmarking Methodology Work Group (BMWG) goals. For example, in [Mor16], "Considerations for Benchmarking Virtual Network Functions and Their Infrastructure" presents relevant statements concerning, for example, utilization of Infrastructure/Resource domains resources; noisy behavior caused by VNFs operations and misbehavior in Infrastructure/Resource domains; long-term tests which represent service continuity guarantees for VNFs; and considerations on how shared resources dependencies may affect consistent benchmark results. All these NFV benchmarking aspects are relevant for VBaaS, as well other current [TOM16] and future recommendations from BMWG.

In OPNFV (Open Platform for NFV) community, projects dedicated to VIM and NFVI tests are being developed, such as vSwitchPerf, Pharos, Functest, among others. Specifically, Yardstick [OPN16] is closely related with VBaaS, because it stands for a common definition of tests to verify the infrastructure compliance when running VNF applications. While VBaaS is still building its concepts, Yardstick already defined requirements, methodologies (including metrics for compute, network and storage domains), besides functional code. The specification of yaml files to define structures like scenarios, contexts and runners, as a Yardstick nomenclature to compose tests, represents associations with VNF-BPs concepts. As Yardstick future work intends to apply tests in compute and storage domains, and store tests results, it is on its way the possible intersection with VBaaS concepts, like VNF Profiles.

3 Virtualization

3.1 Virtualizer library updates

The Virtualizer library, introduced in [D3.4] as Virtualizer3, is the key enabler of a common Sl-Or (UNIFY) interface. As Sl-Or is the most important interface of the UNIFY architecture appearing at different layers, we proposed a common library to be used by all partners for implementing related interfaces. The Virtualizer3 library is an implementation of our Virtualizer YANG model defined in [D3.2a] and further elaborated in [D3.3].

Since [D3.3], based on the proof-of-concept experiments, we have updated the Virtualizer model, summarized as a tree-view in Listing 4.

The main updates are highlighted here:

- Ports have been massively updated, see details in the next subsections:
  - Describing SAPs
  - Describing the Cf-Or interface
  - Describing addresses of NF ports
  - Describing (e.g. matching) capabilities of NF ports
- Nodes and ports can have metadata, which can be used for domain or technology specific data (while handled transparently by other domains), see section 3.1.5.
- The id is a mandatory field
- Resources may have an administrative cost
- The version of the Virtualizer model and library is included

Listing 4: Virtualizer v5: YANG tree

```
module: virtualizer
   +--rw virtualizer
      +--rw id              string
      +--rw name?           string
      +--rw nodes
      |  +--rw node* [id]
      |     +--rw id                 string
      |     +--rw name?              string
      |     +--rw type               string
      |     +--rw ports
      |     |  +--rw port* [id]
      |     |     +--rw id                 string
      |     |     +--rw name?              string
      |     |     +--rw port_type?         string
      |     |     +--rw capability?        string
      |     |     +--rw sap?               string
      |     |     +--rw sap_data
      |     |     |  +--rw technology?     string
      |     |     |  +--rw resources
      |     |     |     +--rw delay?       string
      |     |     |     +--rw bandwidth?   string
      |     |     |     +--rw cost?        string
      |     |     +--rw control
```

```
|   |   |   +--rw controller?      string
|   |   |   +--rw orchestrator?    string
|   |   +--rw addresses
|   |   |   +--rw l2?    string
|   |   |   +--rw l3* [id]
|   |   |   |   +--rw id              string
|   |   |   |   +--rw name?           string
|   |   |   |   +--rw configure?      string
|   |   |   |   +--rw client?         string
|   |   |   |   +--rw requested?      string
|   |   |   |   +--rw provided?       string
|   |   |   +--rw l4?    string
|   |   +--rw metadata* [key]
|   |   |   +--rw key      string
|   |   |   +--rw value?   string
|   +--rw links
|   |   +--rw link* [id]
|   |   |   +--rw id              string
|   |   |   +--rw name?           string
|   |   |   +--rw src?            -->
|   |   |   +--rw dst?            -->
|   |   |   +--rw resources
|   |   |   |   +--rw delay?          string
|   |   |   |   +--rw bandwidth?      string
|   |   |   |   +--rw cost?           string
|   +--rw resources
|   |   +--rw cpu        string
|   |   +--rw mem        string
|   |   +--rw storage    string
|   |   +--rw cost?      string
|   +--rw metadata* [key]
|   |   +--rw key      string
|   |   +--rw value?   string
|   +--rw NF_instances
|   |   +--rw node* [id]
|   |   |   +--rw id              string
|   |   |   +--rw name?           string
|   |   |   +--rw type?           string
|   |   |   +--rw ports
|   |   |   |   +--rw port* [id]
|   |   |   |   |   +--rw id              string
|   |   |   |   |   +--rw name?           string
|   |   |   |   |   +--rw port_type?      string
|   |   |   |   |   +--rw capability?     string
|   |   |   |   |   +--rw sap?            string
|   |   |   |   |   +--rw sap_data
|   |   |   |   |   |   +--rw technology?    string
|   |   |   |   |   |   +--rw resources
|   |   |   |   |   |   |   +--rw delay?         string
|   |   |   |   |   |   |   +--rw bandwidth?     string
|   |   |   |   |   |   |   +--rw cost?          string
|   |   |   |   |   +--rw control
|   |   |   |   |   |   +--rw controller?      string
|   |   |   |   |   |   +--rw orchestrator?    string
|   |   |   |   |   +--rw addresses
|   |   |   |   |   |   +--rw l2?    string
|   |   |   |   |   |   +--rw l3* [id]
|   |   |   |   |   |   |   +--rw id              string
|   |   |   |   |   |   |   +--rw name?           string
|   |   |   |   |   |   |   +--rw configure?      string
```

```
84          |       |       |       |   |  +--rw client?          string
85          |       |       |       |   |  +--rw requested?       string
86          |       |       |       |   |  +--rw provided?        string
87          |       |       |       |   +--rw l4?     string
88          |       |       |       +--rw metadata* [key]
89          |       |       |          +--rw key      string
90          |       |       |          +--rw value?   string
91          |       |       +--rw links
92          |       |       |  +--rw link* [id]
93          |       |       |     +--rw id            string
94          |       |       |     +--rw name?         string
95          |       |       |     +--rw src?          -->
96          |       |       |     +--rw dst?          -->
97          |       |       |     +--rw resources
98          |       |       |        +--rw delay?     string
99          |       |       |        +--rw bandwidth? string
100         |       |       |        +--rw cost?      string
101         |       |       +--rw resources
102         |       |       |  +--rw cpu          string
103         |       |       |  +--rw mem          string
104         |       |       |  +--rw storage      string
105         |       |       |  +--rw cost?        string
106         |       |       +--rw metadata* [key]
107         |       |          +--rw key      string
108         |       |          +--rw value?   string
109         |       +--rw capabilities
110         |       |  +--rw supported_NFs
111         |       |     +--rw node* [id]
112         |       |        +--rw id             string
113         |       |        +--rw name?          string
114         |       |        +--rw type?          string
115         |       |        +--rw ports
116         |       |        |  +--rw port* [id]
117         |       |        |     +--rw id             string
118         |       |        |     +--rw name?          string
119         |       |        |     +--rw port_type?     string
120         |       |        |     +--rw capability?    string
121         |       |        |     +--rw sap?           string
122         |       |        |     +--rw sap_data
123         |       |        |     |  +--rw technology?    string
124         |       |        |     |  +--rw resources
125         |       |        |     |     +--rw delay?      string
126         |       |        |     |     +--rw bandwidth?  string
127         |       |        |     |     +--rw cost?       string
128         |       |        |     +--rw control
129         |       |        |     |  +--rw controller?    string
130         |       |        |     |  +--rw orchestrator?  string
131         |       |        |     +--rw addresses
132         |       |        |     |  +--rw l2?    string
133         |       |        |     |  +--rw l3* [id]
134         |       |        |     |     +--rw id          string
135         |       |        |     |     +--rw name?       string
136         |       |        |     |     +--rw configure?  string
137         |       |        |     |     +--rw client?     string
138         |       |        |     |     +--rw requested?  string
139         |       |        |     |     +--rw provided?   string
140         |       |        |     |  +--rw l4?    string
141         |       |        |     +--rw metadata* [key]
142         |       |        |        +--rw key      string
143         |       |        |        +--rw value?   string
```

```
144         |       |           +--rw links
145         |       |           |  +--rw link* [id]
146         |       |           |     +--rw id              string
147         |       |           |     +--rw name?           string
148         |       |           |     +--rw src?            -->
149         |       |           |     +--rw dst?            -->
150         |       |           |     +--rw resources
151         |       |           |        +--rw delay?       string
152         |       |           |        +--rw bandwidth?   string
153         |       |           |        +--rw cost?        string
154         |       |           +--rw resources
155         |       |           |  +--rw cpu          string
156         |       |           |  +--rw mem          string
157         |       |           |  +--rw storage      string
158         |       |           |  +--rw cost?        string
159         |       |           +--rw metadata* [key]
160         |       |              +--rw key        string
161         |       |              +--rw value?     string
162         |       +--rw flowtable
163         |          +--rw flowentry* [id]
164         |             +--rw id              string
165         |             +--rw name?           string
166         |             +--rw priority?       string
167         |             +--rw port            -->
168         |             +--rw match           string
169         |             +--rw action          string
170         |             +--rw out?            -->
171         |             +--rw resources
172         |                +--rw delay?       string
173         |                +--rw bandwidth?   string
174         |                +--rw cost?        string
175         +--rw links
176         |  +--rw link* [id]
177         |     +--rw id              string
178         |     +--rw name?           string
179         |     +--rw src?            -->
180         |     +--rw dst?            -->
181         |     +--rw resources
182         |        +--rw delay?       string
183         |        +--rw bandwidth?   string
184         |        +--rw cost?        string
185         +--rw metadata* [key]
186         |  +--rw key        string
187         |  +--rw value?     string
188         +--rw version?      string
```

3.1.1 SAP ports

The Interdomain connection identifier (SAP) is demonstrated in Listing 5. If SAPs in different domains report the same ID, they are assumed to be directly interconnected.

Listing 5: SAP identifier: xml view

```
1 <port>
2   <id>01</id>
3   <name>Simple inter-domain SAP</name>
4   <port_type>port-sap</port_type>
5   <sap>SAP14</sap>
6 </port>
```

A SAP port resources example is shown in Listing 6. The upper layer orchestrator (stitching the two domains together) will convert the two SAP port resources to a link with resources between these ports.

Listing 6: SAP resources: xml view

```
<port>
  <id>02</id>
  <name>Inter-domain SAP with resources</name>
  <port_type>port-sap</port_type>
  <sap>SAP15</sap>
  <sap_data>
    <resources>
      <delay>3 ms</delay>
      <bandwidth>100 MB/s</bandwidth>
      <cost>10</cost>
    </resources>
  </sap_data>
</port>
```

Listing 7 shows how to represent the technology (e.g. IEEE802.1q, MPLS, ...) used between domains for the Unify tagging.

Listing 7: Unify tagging technology: xml view

```
<port>
  <id>13</id>
  <name>Inter-domain SAP port with Unify-tag technology specified </name>
  <port_type>port-sap</port_type>
  <sap>SAP100</sap>
  <sap_data>
    <!-- note: the format below is a Python list : -->
    <technology >['IEEE802.1q']</technology>
  </sap_data>
</port>
```

The full protocol stack of the SAP, including the Unify tag is shown in the example of Listing 8.

Listing 8: SAP protocol stack: xml view

```
<port>
  <id>14</id>
  <name>Inter-domain SAP port with full protocol stack</name>
  <port_type>port-sap</port_type>
  <sap>SAP101</sap>
  <sap_data>
    <!-- note: the format below is a Python list , where the elements are tuples : -->
    <technology >[('IEEE802.1q', '0x00c'), ('MPLS', 70), ('IEEE802.1q')]</technology>
  </sap_data>
</port>
```

3.1.2 Port addresses

Listing 9 demonstrates how a public L3 (IP) address can be requested, then Listing 10 how it is provided. If a public IP is requested, the port is assumed to be connected to public routable network, like INTERNET. In this case, there is no UNIFY forwarding configured for this port!

Listing 9: Public IP address request: xml view

```
<port>
  <id>03a</id>
  <name>Public IP port to be pushed to NF via DHCP (Step 1: request)</name>
  <port_type>port-sap</port_type>
  <sap>INTERNET</sap>
  <addresses>
    <l3>
      <id>addr1</id>
      <!-- address is to be pushed to NF: -->
      <configure>True</configure>
      <!-- client uses DHCP to receive address: -->
      <client>dhcp-client</client>
      <requested> public </requested>
    </l3>
  </addresses>
</port>
```

Listing 10: Public IP address response: xml view

```
<port>
  <id>03b</id>
  <name>Public IP port to be pushed to NF via DHCP (Step 2: response)</name>
  <port_type>port-sap</port_type>
  <sap>INTERNET</sap>
  <addresses>
    <l3>
      <id>addr1</id>
      <configure>True</configure>
      <client>dhcp-client</client>
      <requested> public </requested>
      <!-- the address given: -->
      <provided>8.8.8.8</provided>
    </l3>
  </addresses>
</port>
```

Listing 11 shows how a private IP address can be requested, then Listing 12 how it is provided. This address is internal to the NF, the forwarding will be configured by UNIFY.

Listing 11: Private IP address request: xml view

```
<port>
  <id>04a</id>
  <name>Private IP (Step 1: resquest)</name>
  <port_type>port-sap</port_type>
  <sap>SAP22</sap>
  <addresses>
    <l3>
      <id>addr1</id>
      <configure> False </configure>
      <requested>192.168.0.1/24</requested>
    </l3>
  </addresses>
</port>
```

Listing 12: Private IP address response: xml view

```
<port>
  <id>04b</id>
```

```
  <name>Private IP (Step 2: response)</name>
  <port_type>port-sap</port_type>
  <sap>SAP22</sap>
  <addresses>
    <l3>
      <id>addr1</id>
      <configure> False </configure>
      <requested >192.168.0.1/24</ requested>
      <provided >192.168.0.1/24</ provided>
    </l3>
  </addresses>
</port>
```

The next example pair in Listing 13 and Listing 14 demonstrates the case, when only a few listed L4 ports (TCP or UDP) are to be made accessible. This case can be realized by the infrastructure by e.g. with NAT.

Listing 13: Port list request: xml view

```
<port>
  <id>05a</id>
  <name>L4 port (Step 1: resquest )</name>
  <port_type>port-sap</port_type>
  <sap>INTERNET</sap>
  <addresses>
    <!-- note: the format below is a Python set: -->
    <l4 >(' tcp /22', ' tcp /80')</ l4>
  </addresses>
</port>
```

Listing 14: Port list response: xml view

```
<port>
  <id>05b</id>
  <name>L4 port (Step 2: response)</name>
  <port_type>port-sap</port_type>
  <sap>INTERNET</sap>
  <addresses>
    <!-- note: the format below is a Python dict, where the values are tuples: -->
    <l4 >{' tcp /22': ('192.168.1.100', '1001') , ' tcp /80': ('192.168.1.150', '2001') }</ l4>
  </addresses>
</port>
```

Listing 15 and Listing 16 shows requesting both a public and a private IP for the same port. An example for such a request can be requesting a VM behind a NAT, with a private address assigned to the VM interface and a public address where it is accessible from far.

Listing 15: Private and public IP address request: xml view

```
<port>
  <id>06a</id>
  <name>Private and public IP port (Step 1: resquest )</name>
  <port_type>port-sap</port_type>
  <sap>INTERNET</sap>
  <addresses>
    <l3>
      <id>addr1</id>
      <configure>True</configure>
      <requested> private </requested>
    </l3>
```

```
    <l3>
      <id>addr2</id>
      <configure> False </configure>
      <requested> public </requested>
    </l3>
  </addresses>
</port>
```

Listing 16: Private and public IP address response: xml view

```
<port>
  <id>06b</id>
  <name>Private and public IP port (Step 2: response)</name>
  <port_type>port-sap</port_type>
  <sap>INTERNET</sap>
  <addresses>
    <l3>
      <id>addr1</id>
      <configure>True</configure>
      <requested> private </requested>
      <provided >192.168.1.5/24</ provided>
    </l3>
    <l3>
      <id>addr2</id>
      <configure> False </configure>
      <requested> public </requested>
      <provided >8.8.8.10</ provided>
    </l3>
  </addresses>
</port>
```

3.1.3 Port capabilities

The next examples in Listing 17 show how to describe match and action capabilities associated with the ports, e.g., match=port,tag,ip,tcp,udp,mpls,of1.0. Where port means port based forwarding; tag means Unify abstract tagging; ip means IP address matching etc.

Listing 17: Capabilities of ports: xml view

```
<ports>
  <port>
    <id>07</id>
    <name>Abstract Unify port with capabilities </name>
    <port_type>port-abstract</port_type>
    <!-- OpenFlow 1.0 match and action capable port: -->
    < capability >of1.0</ capability >
  </port>
  <port>
    <id>08</id>
    <name>Abstract Unify port with capabilities </name>
    <port_type>port-abstract</port_type>
    <!-- Only port match/action : -->
    < capability >port</ capability >
  </port>
  <port>
    <id>09</id>
    <name>Abstract Unify port with capabilities </name>
    <port_type>port-abstract</port_type>
```

```
20      <!-- Unify abstract tag capable port: -->
21      <capability>tag</capability>
22    </port>
23    <port>
24      <id>10</id>
25      <name>Abstract Unify port with capabilities</name>
26      <port_type>port-abstract</port_type>
27      <!-- Capabilitites listed: -->
28      <capability>port,tag,ip,tcp,udp,mpls</capability>
29    </port>
30  </ports>
```

3.1.4 Cf-Or interface

Listing 18 demonstrates a port allocated as the Cf-Or interface. The control tag is used to connect this port to a UNIFY orchestrator's Cf-Or reference point. The model supports controller → orchestrator or orchestrator → controller connection establishment as well. Which fields of the control tag are filled in at the request depend on the handshake direction between the controller and the orchestrator. In the end both the controller and orchestrator will be filled in.

Listing 18: Cf-Or interface: xml view

```
1  <port>
2    <id>11</id>
3    <name>Port with Cf-Or interface</name>
4    <port_type>port-abstract</port_type>
5    <control>
6      <!-- This is the connection point at the NF (IP address is not filled in, it can be obtained from the public port): -->
7      <controller>http://*:8080/cf-or/</controller>
8      <!-- This is the connection point at the orchestrator: -->
9      <orchestrator>http://192.168.1.100:8080/cf-or/</orchestrator>
10   </control>
11 </port>
```

Example of Listing 19 demonstrates a port with a public IP, as well being the Cf-Or interface.

Listing 19: Cf-Or interface with public IP: xml view

```
1  <port>
2    <id>12</id>
3    <name>Public port with Cf-Or interface</name>
4    <port_type>port-sap</port_type>
5    <sap>INTERNET</sap>
6    <control>
7      <controller>http://*:8080/cf-or/</controller>
8      <orchestrator>http://192.168.1.100:8080/cf-or/</orchestrator>
9    </control>
10   <addresses>
11     <l3>
12       <id>addr1</id>
13       <configure>True</configure>
14       <client>dhcp-client</client>
15       <requested>public</requested>
16       <provided>8.8.8.8</provided>
17     </l3>
18   </addresses>
19 </port>
```

3.1.5 Metadata

Listing 20 gives a complex example how the metadata can be used to encode technology specific SAP information, e.g. VxLAN.

Listing 20: Metadata (e.g. VxLAN): xml view

```
<port>
  <id>15</id>
  <name>experimental: SAP port with VXLAN behind NAT</name>
  <port_type>port-sap</port_type>
  <sap>INTERNET</sap>
  <sap_data>
    <technology >(' vxlan ') </technology>
  </sap_data>
  <addresses>
    <l3>
      <id>addr1</id>
      <requested> private </requested>
      <provided >192.168.1.5/24</ provided>
    </l3>
    <l3>
      <id>addr2</id>
      <requested> public </requested>
      <provided >8.8.8.10</ provided>
    </l3>
  </addresses>
  <metadata>
    <key> vxlan_key </key>
    <value>11</value>
  </metadata>
  <metadata>
    <key> vxlan_localip </key>
    <value >10.1.2.10</ value>
  </metadata>
  <metadata>
    <key> vxlan_remoteip </key>
    <value >10.0.2.108</ value>
  </metadata>
</port>
```

3.2 Different choices for virtualization

Virtualization of the infrastructure gives network providers the freedom to abstract their infrastructure which has several advantages: i) less information is exposed which is useful for hiding business secrets and for avoiding attacks, ii) changes in the infrastructure do not need to be exposed which leads to lower information exchange, iii) users are not required to be concerned about the exact placement of the requested NFs and iv) the provider can optimize its resource usage. However, the abstract view of the infrastructure might lead to i) increased requests rejection and ii) reduced freedom for users to map their requested NFs. Therefore, a main challenge is to define a good virtualization taking into account different perspectives. We identify different trade-offs and trends with respect to the infrastructure dynamics, information exposure and the abstraction level achieved by different virtualizations. We investigate a partitioning algorithm to create different virtualized views of the infrastructure and evaluate these views with respect to stability and embedding efficiency. Then based on the results, we propose a simple heuristic which enables the network provider to select a suitable virtualized topology.

3.2.1 Construction of the virtualizer

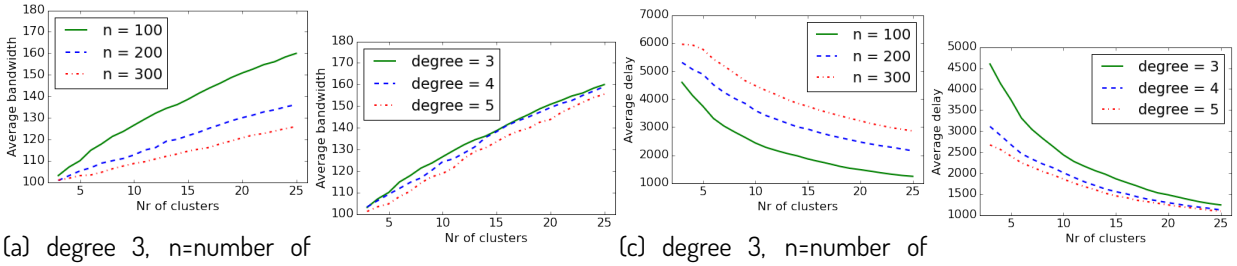
Problem statement. Given a physical infrastructure modeled as an undirected graph $G_p = (N_p, L_p)$ composed of nodes (N_p) connected via links (L_p), assuming each node has certain capacity in terms of computation (C), memory (M) and storage (S) and links have delay (D) and capacity in terms of bandwidth (BW), we search for an optimal virtualization of the resources. However, the metrics for which the virtualization should be optimized depend on the two aforementioned perspectives, i.e., user and network provider. Therefore, in order to construct an optimal virtualizer considering different perspectives, several factors should be taken into account: i) which links/nodes to expose, ii) how to calculate the characteristics (e.g., compute, bandwidth) of the abstract nodes and links, iii) which algorithm to use to create a partition of the nodes, in which each set (i.e. cluster) of nodes results in an abstract node, etc.

Proposed approach. In this section, we explain the approach we used to construct a virtualiziton of an infrastructure: i) we use a partitioning algorithm on the infrastructure, $G_p = (N_p, L_p)$, ii) clusters of nodes which are the outcome of the partitioning become abstract nodes in the new topology, iii) two abstract nodes get an abstract link if any two nodes, one from each cluster, are connected in the infrastructure, iv) the resources of the infrastructure are combined into the abstract nodes/links (i.e. C, M and S are aggregated, links get assigned minimal bandwidth BW, and maximal delay. This means that the shortest paths between nodes of one cluster to the other one are calculated. The minimum bandwidth of the links of all the paths is exposed in the virtualizer and the maximum sum of the delays of the links of the paths is considered as the delay of the abstract link) and v) all the abstract nodes/links get exposed to the user with their resources using the BiS-BiS virtualization model.

Community partitioning. It is expected that abstract nodes containing closely linked infrastructure nodes to run resource-heavy service requests with better performance than if they were poorly connected. If multiple infrastructure nodes are needed to handle a service request, them being closely connected will improve the performance of the request. Also, the closer that interacting nodes are connected, the less other nodes have to aid in their communication, meaning less used resources from the other nodes. Therefore the Louvain method [Blo08] seems like a promising approach for the partitioning. This method divides the nodes of a network into communities (i.e. clusters) so that the nodes inside a community are well-connected. In [Blo08], the authors define the metric modularity, which is a scalar that increases when the intra-cluster connectivity becomes more dense, compared to the inter-cluster links. This method iterates over possible partitions by moving nodes from their own cluster into the cluster of one of their neighbors, in order to optimize the modularity. The Louvain method returns a set of partitions that resemble local maxima in the modularity. The amount of communities for each partition is not predictable. As we need different virtualizers, we extend the Louvain method by starting from the lowest-level partition (i.e. the one with the most communities) and merging those two communities that resemble the highest modularity gain, on the condition that these communities are in the same cluster in the higher-level partitions. We repeat this until only 1 cluster remains. This way we can request a certain number of communities in a partition. This enables us to consider more virtualizers to identify trade-offs between different metrics such as between the abstraction level of the virtualizer and the acceptance rate of the embedding process.

3.2.2 Experimental results

As mentioned before, finding/defining an optimal virtualizer is challenging. The reason is that different perspectives and factors should be taken into account. Therefore we took a different approach. We first evaluated the generated virtualizers with respect to: i) average delay and bandwidth of the virtualizer links, ii) stability: indication of the amount of changes in the virtualizer upon dynamics in the infrastructure, iii) similarity: indication of the similarity of the virtualizer



(a) degree 3, n=number of nodes (b) 100 nodes, different degreenodes (c) degree 3, n=number of (d) 100 nodes, different degree

Figure 15: Average bandwidth and delay evaluation

and the infrastructure and iv) acceptance rate: ratio between the accepted requests and all the requests. Furthermore, we evaluated the impact of topology characteristics (such as degree and number of nodes) on these metrics. Then based on the identified trade-offs, we proposed a heuristic to select a suitable virtualizer.

Simulation setup. We generated random regular networks with different size and degree as infrastructure. The capacity of nodes (in terms of C, M, S) and links (in terms of BW and D) are numbers uniformly distributed between 100 and 300. The generated virtualizers are based on the partitions resulting from our extension of the Louvain method, where C, M and S of an abstract node are calculated as the sum of the resources over all nodes within that abstract node. The service requests are directed graphs (NFs as nodes and connections as links) which are generated randomly and each pair of nodes is connected with probability 0.5. The resource demands of the NFs and the links are numbers uniformly distributed between 1 and 50. The requests arrive in a Poisson process with average rate of 4 requests per 100 time units and have exponentially distributed lifetimes with an average of μ = 1000 time units. Each request contains 5 to 10 NFs. Each scenario is iterated 20 times and results are averaged.

Simulation results. First we evaluate the impact of different virtualizers on the calculated bandwidth and delay of the virtualizer links. These metrics influence the embedding process performance. The bandwidth of a virtualizer link (l_v) is defined as:

$$BW(l_v) = min(BW(l_p), \forall l_p \in P, \forall P \in Paths)$$

$Paths$ includes all paths (see further) that connect any two nodes, one from each cluster connected by the abstract link. The delay of a link is defined as:

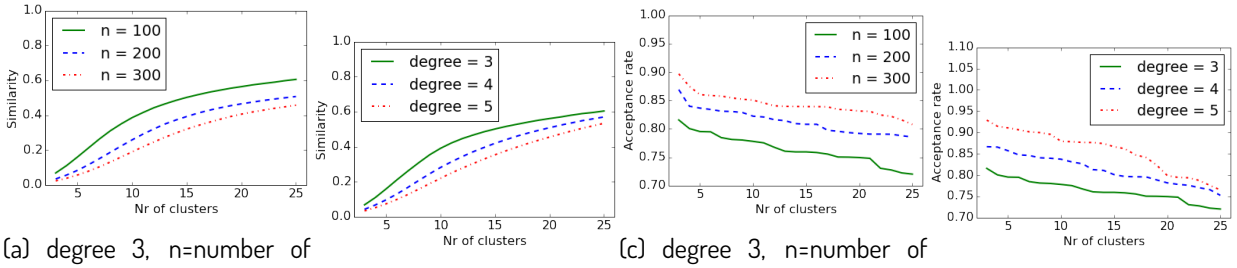
$$D(l_v) = max(\sum_{\forall l_p \in P} D(l_p), \forall P \in Paths)$$

There are different algorithms to calculate these paths e.g., shortest path, or least cost path with different cost metrics. We considered shortest path algorithm for these evaluations. Fig. 15 reports the averaged bandwidth and delay over all links in a virtualizer. As we see, increasing the abstraction results in a lower bandwidth (Fig. 15a, 15b) and a higher delay (Fig. 15c, 15d). When the number of clusters is constant, we observe that larger networks have worse bandwidth and delay while lower delay in networks with higher degree is achieved. The impact of degree on bandwidth is negligible.

Next we define a metric referred to as 'similarity'. This metric indicates the similarity of the virtualizer and the original network. We believe this is an important metric as providers prefer to expose information of the infrastructure as little as possible. This metric is defined as:

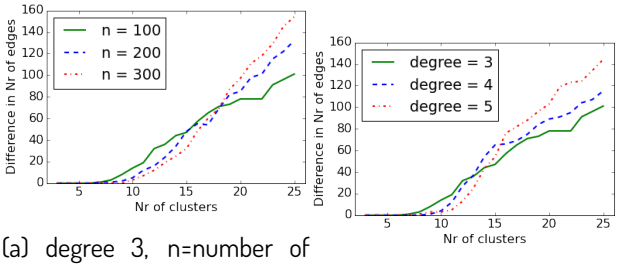
$$E(V, G) = \frac{1}{2}(\frac{\|N_v\|}{\|N_p\|} + \frac{\|L_v\|}{\|L_i\|})$$

N_v is the set of clusters, N_p is the set of nodes, L_v is the set of tuples $(k_1; k_2)$ with k_1 and k_2 directly connected clusters and L_i is the set of infrastructure links that are connecting clusters. The more nodes (links) that are combined in (between) clusters, the more difficult it is to reconstruct the original topology and the lower E will be. This is illustrated in Fig. 16a, where the similarity decreases when the average cluster size increases (decreasing number of clusters)



(a) degree 3, n=number of nodes (b) 100 nodes, different degreenodes (c) degree 3, n=number of (d) 100 nodes, different degree

Figure 16: Similarity and acceptance rate evaluation



(a) degree 3, n=number of nodes (b) 100 nodes-different degree

Figure 17: Stability evaluation

and when the number of clusters is kept constant, while the number of nodes increases. Fig. 16b indicates that the similarity decreases when the degree increases, as the ratio connected clusters to inter-cluster links decreases.

In the next experiment, we evaluate the impact of different virutalizer on the performance of the embedding algorithm in terms of acceptance ratio. We extended the embedding algorithm explained in [D3.2] to support two-level embedding. First a request is mapped to the virtualizer. If this first level is successful, the same algorithm is used to map the NFs (links) assigned to a cluster to the nodes (links) within that cluster. For the mapping of the service links to inter-cluster links, we search for the shortest path which fulfills the link demand between the two nodes hosting the two mapped NFs. The acceptance rate for different virtualizers and networks with different size and degree is illustrated in Fig. 16c and 16d. The increase in the number of nodes and degree improves the acceptance rate of the embedding process. Additionally, we observe that a higher abstraction results in better performance of the embedding. This can be explained by the fact that if the virtualizer is close to a single BiS-BiS, the first-level embedding is trivial and the second-level embedding is very close to the embedding on the actual infrastructure. This seems to conflict with the trend that bandwidth and delay get worse when the abstraction increases. However, many different factors should be taken into account e.g.: how bandwidth and delay are defined (the approach that is used for calculation of the resources exposed by the virtualizer), how the embedding is performed and what the requests characteristics (e.g., arrival rate, life time, etc.) are.

Finally, we evaluate the impact of different virtualizers on the stability of the information. To this end, we calculated the number of changes made in the virtualized topology caused by 10% link failures in the original infrastructure. These changes consist of adding and removing links in the virtualizer. The number of changes in the virtualizers are reported in Fig. 17. The trend of increasing instability for increasing number of clusters becomes steeper for larger networks and for a higher degree.

3.2.3 Heuristic algorithm for virtualization selection

The experimental results indicated the impact of abstraction level on different metrics. We propose a heuristic to select a suitable virtualizer to have a balance between the amount of information revealed and the chance of accepting the requests. In this heuristic we define a cost function which is composed of the following factors: i) similarity (E), ii) average delay of links (D), iii) average bandwidth of links (BW), iv) acceptance rate of requests (A) and v) stability (S), which is the number of changes upon failure. The cost function is defined as:

$$C(v) = \alpha \cdot E_v + \beta \cdot D_v + \delta \cdot S_v + \gamma / BW_v + \kappa / A_v$$

The parameters α, β, δ, γ and κ are used to tune the impact of each factor. If there are multiple ways of virtualizing the original infrastructure (e.g., by using different partitioning algorithms), we first calculate the cost of each vritualizer and the minimum cost virtualizer is sent to the user. In the cost function, only E, D and BW can be calculated based on the infrastructure, independent of the requests. For calculation of other parameters, i.e., A and S, the provider should perform profiling of: i) the requests and ii) the network dynamics. The pseudo code of this approach is shown in Algorithm 2.

Algorithm 2: VirtualizerSelection pseudo code

```
1  VirtualizerSelection(Infrastructure, Virtualizers)
   Data: Infrastructure, Virtualizers
   Result: minimum cost virtualizer
2  Cost=[];
3  for a ∈ Virtualizers do
4      E_a = similarity between infrastructure and a;
5      D_a = average (delay edges in a);
6      BW_a = average (bandwidth edges in a);
7      A_a = acceptance rate of a based on requests profiling;
8      S_a = number of changes in a upon certain failures;
9      Cost(a) = α·E_v + β·D_v + δ/S_v + γ/BW_v + κ/A_v;
10 select minimum(Cost);
```

A more suitable virtualizer can be selected if more information about the requests are available, e.g., if more delay-sensitive applications are requested, the parameters in the cost function can be tuned to allow for a lower average delay. Similarly a virtualizer with a higher average bandwidth can be obtained when bandwidth-intensive applications are requested.

4 Use-case integration

4.1 Elastic Router integration

In this section, the implementation of the elastic router use-case is explained more in detail, mainly including the integration with other work packages, such as the monitoring and troubleshooting tools developed in WP4 and the Universal Node of WP5. The virtualizer library developed in WP3 describes the elastic router NF-FG. This builds further upon the basic outline of the elastic router use-case, described in [D3.3], Section 2.5.1.

4.1.1 Elastic router: Deployment process

We first briefly recap the elastic router use-case, as implemented in the Unify project. In its most high-level description (the Service Graph in Figure 18), the elastic router is a service, routing traffic between 4 ports (SAP 1-4). It has an additional external interface which can be used for configuration (eg. set the routing tables).

The Service Graph (SG) is translated to an NF-FG by the Service Layer. This NF-FG is a more practical description of the service, describing the VNFs, SAPs and links to be deployed. Next to this, the SG has a set of requirements in the form of a Service Level Agreement (SLA). In the Service Layer, the SLA is translated into specific observable KPIs. These KPIs are then expressed as MEASURE annotations to the derived NF-FG [5] . So in addition to the VNFs, SAPs and links, the NF-FG defines a set of monitoring rules and alarms, which can be used for example to trigger a scaling action of the elastic router.

The global orchestrator in Figure 18 deploys the NF-FG on the available infrastructure. In this integrated prototype, both service layer and global orchestrator are implemented in the ESCAPE orchestration framework [D3.4]. In this use-case, the infrastructure nodes are Universal Nodes (UN), described in [D5.5]. Each UN has a local orchestrator, which deploys the received NF-FG locally. The global orchestrator (Escape) can map to different infrastructure nodes, also in other domains than the UN, as described in Annex A.

The deployment of the elastic router in the Unfiy framework consists out of four distinct stages, as illustrated in Figure 18:

1. At initial service deployment, a Service Graph (SG) is sent to the Service Layer. This SG is a high-level description of the elastic router, describing the external interfaces connected to the service [6].

2. The Service Layer generates the NF-FG from the SG with information fetched from the NF-IB, where the decomposed topology of the elastic router is stored.

3. The NF-FG is passed to a (global) orchestrator which determines the best choice of infrastructure mapping, deciding where the VNFs will be deployed. The local orchestrator on the infrastructure node will deploy the VNFs trigger the installation of the monitoring functions defined in the MEASURE part of the NF-FG. The monitoring controller (called MMP in the UN) that installs the required monitoring functions is further detailed in 4.1.4.

4. At the service runtime, elastic scaling is possible. A condition derived from the MEASURE annotations triggers the generation of a new, scaled version of the NF-FG in the elastic router Control NF. Via the Cf-Or interface, the updated NF-FGs is then sent to the (global) orchestrator to be deployed. Since an updated NF-FG might contain

[5] The MEASURE monitoring language is introduced in [M4.3] and further detailed in [D4.3], Section 4.2.
[6] The Escape JSON SG model is documented in [D3.4] and in https://sb.tmit.bme.hu/mediawiki/index.php/ESCAPE

VNFs which can be orchestrated among different infrastructure nodes, the global orchestrator is the endpoint for the Cf-Or interface.

For further details, the reader is referred to [D3.3], Section 2.5.1, where the elastic router is more elaborately described.

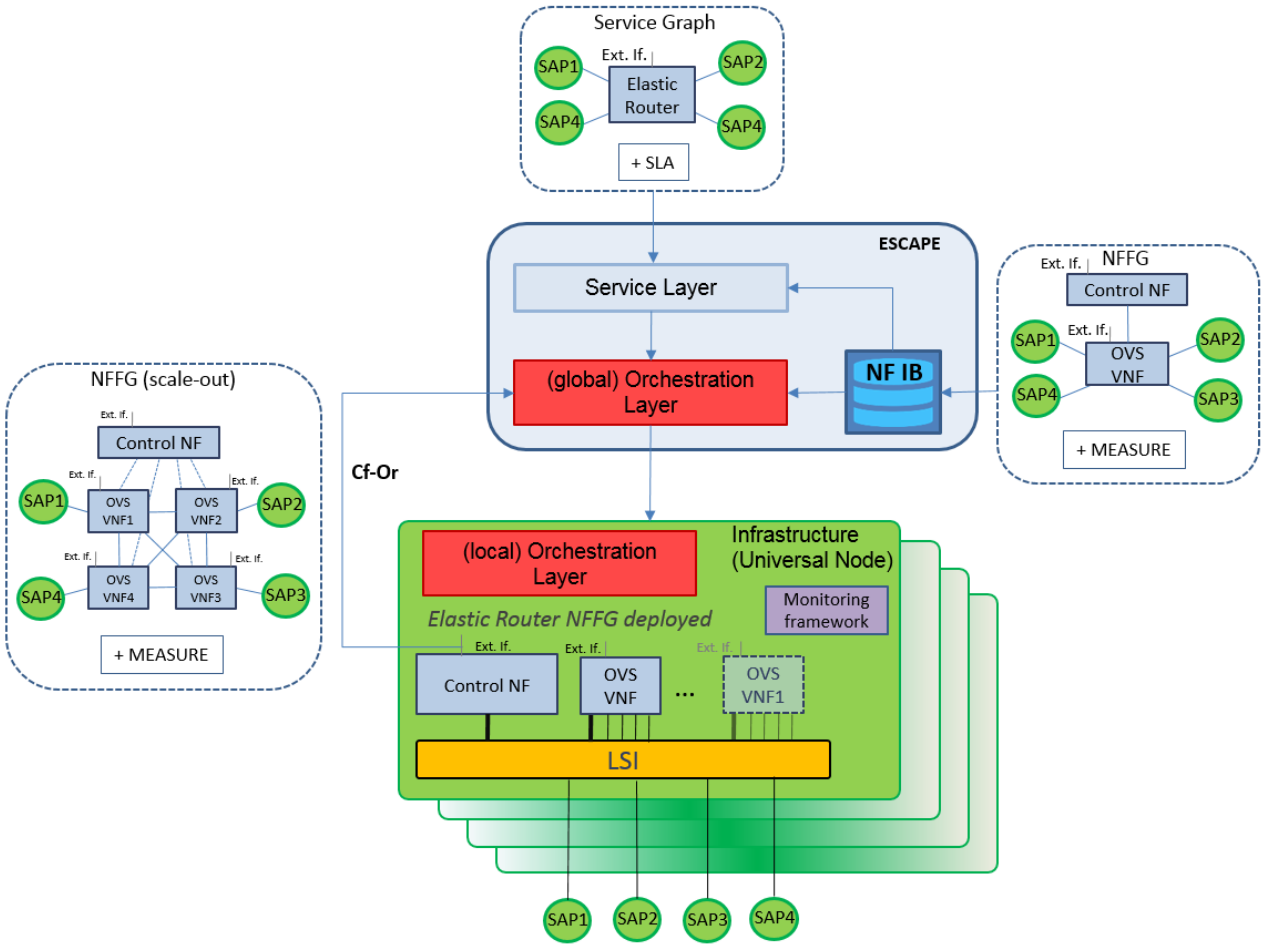


Figure 18: Deployment of the elastic router in the Unify framework.

4.1.2 Elastic Router Decomposition

The NF-FGs used to deploy the elastic router are shown in Figure 18. The NF-FG stored in the NF-IB describes the typical architecture of a software router, decomposed into:

1. The **control plane** is implemented by a dedicated Control NF using an Openflow controller (Ryu) with an extra layer on top of it, which holds the scaling logic. This logic will generate the updated NF-FG to be sent via the Cf-Or. It will also subscribe to the message bus of the monitoring framework to receive published alarms, as derived from the MEASURE annotations. This Cf-Or and message bus connection is done via an external interface, not routed via the LSI (Logical Switch Instance of the infrastructure).

2. The **data plane** functionality is implemented by an OpenVswitch instance (OVS VNF), which flow-entries are set by the Control NF. The SAPs are connected here and all traffic is routed through this OVS VNF. Every OVS VNF also has a control interface that connects the OVS to the OpenFlow Controller in the Control NF. This control

network and the SAP connections are routed via the LSI. Additionally, an external interface is configured on this VNF which can be used to troubleshoot a running OVS instance (not routed via the LSI).

The VNFs deployed as part of the service can also be modified during the service lifetime. This is shown in Figure 18, where the Cf-Or interface is used to define more OVS VNFs to be deployed, in an updated NF-FG (scale-out action). The virtualizer library, as documented in Section 3.1, provides the necessary functionality to describe all needed features of the elastic router NF-FGs:

- The IP addresses of the control interfaces of every VNF of the elastic router service must be specified. This is needed in order to correctly set-up the control network between the Control NF and every OVS VNF. This is documented in Section 3.1.2 by requesting a private IP address for a certain VNF port.
- The Cf-Or interface is the only way for the Control NF to communicate any scaling actions to the orchestrator. By using the control tag, as described in 3.1.4, the Control NF can be configured to contact the orchestrator at the address and port specified in the orchestrator field.
- Every VNF can also be configured with a public available port. This is illustrated in 3.1.2 by requesting a public IP address+port for a certain VNF port. For instance, the public IPs of the OVS VNFs are used in this use-case to allow SSH login for debugging and troubleshooting purposes. For the Control NF, a public port can be used to expose a user interface and allows the receiving of alarms generated by the monitoring framework as well as connection to the Cf-Or interface.
- Additional service or VNF specific parameters can be configured via the virtualizer library using meta-data tags, described in 3.1.5. In the elastic router use-case, these meta-data tags are translated by the UN orchestrator to Docker environment variables, used for example to specify the name or datapath-id of a deployed OVS VNF.

4.1.3 Elastic router: Scaling strategy

In this use-case, the conditions for scaling are assessed relative to defined thresholds and are based on the combined inputs from monitoring functions such as RAMON and CAdvisor.

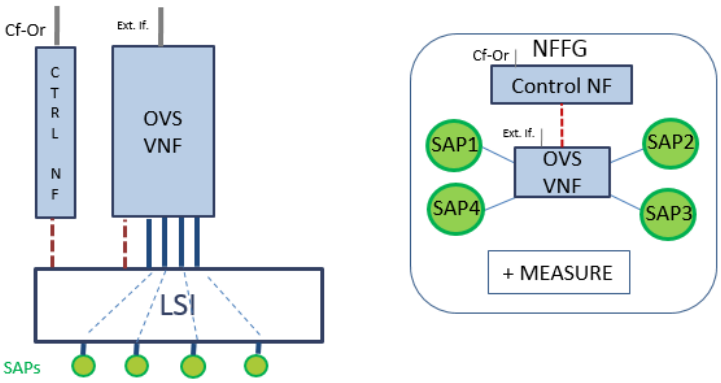
As scaling strategy, the elastic router applies a make-before-break strategy, i.e. to intermediately deploy the scaled topology next to the original one. This allows reliable implementation and state migration, which means that the service is not interrupted during the scaling action. The original non-scaled VNFs need no special re-configuration during scaling, they only get removed once the state migration is complete. On the downside, this scaling strategy is not the most resource optimal, as discussed in [D3.2], Section 5.2.

As seen in Figure 19, multiple updated NF-FGs are generated by the Control NF during a scale-out action:

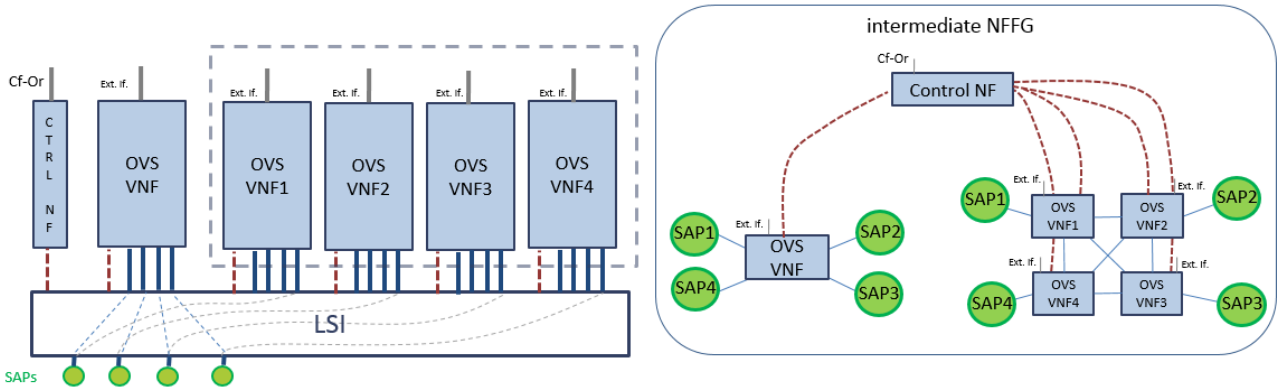
1. The initially deployed NF-FG is used as a starting point. It has a Control NF and one OVS VNF to which 4 SAPs are connected. This is shown in Figure 19a.
2. During a scale action, the Control NF first creates an intermediate NF-FG, in which the scaled out/in topology of the elastic router (more or less OVS VNFs) is created next to the originally deployed OVS VNF, as seen in Figure 19b. Any state (routing tables) is then correctly translated and migrated from the original OVS VNF to the new ones. The SAPs also get routed to the newly created OVS VNFs, but these flowrules have a lower priority so the traffic is till passing through the original OVS VNF.

3. The old flowrules are deleted, so traffic is now re-routed to the new OVS VNFs. Once state migration is complete, the old OVS VNF is deleted. The final NF-FG is shown in Figure 19c.

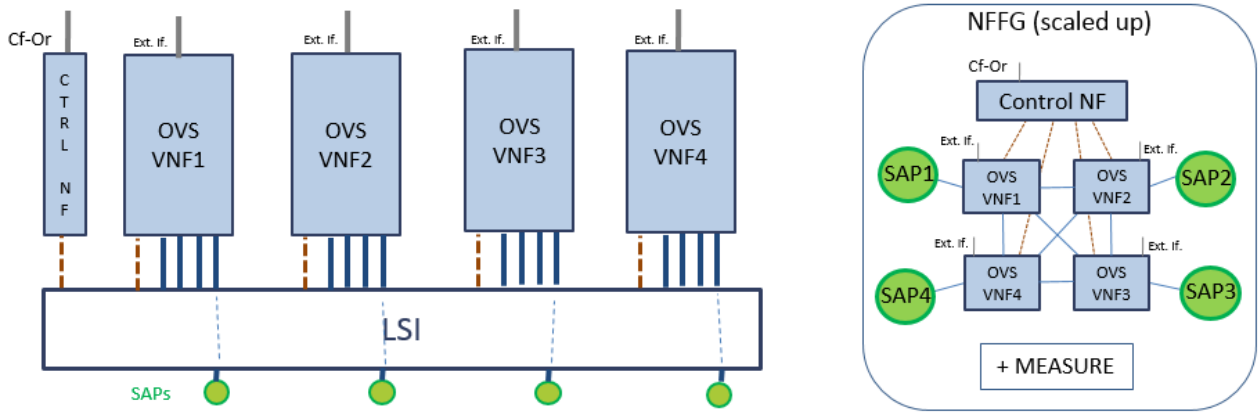
The scale-in action of the elastic router follows the opposite flow, from Figure 19c to 19a.



(a) Initial, non-scaled version of the elastic router NF-FG.



(b) Intermediate elastic router NF-FG, during the scaling out action. OVS VNF1-4 are concurrently deployed.



(c) Final, scaled-out version of the elastic router NF-FG (the original OVS VNF is removed).

Figure 19: Different NF-FGs involved in the scaling out action of the elastic router. The SAPs are re-routed and the orifinal OVS-VNF is removed.

4.1.4 Elastic router: Integrated prototype architecture

The integrated prototype combines different tools and features from WP3, WP4 and WP5. The elastic router use-case is used to demonstrate the deployment process of a NF-FG in the Unify framework, together with monitoring and troubleshooting features. During the service lifetime, monitor tools will trigger the Control NF to scale-out/in more or less OVS VNFs. Next to this, troubleshooting tools enable efficient debugging of the elastic-router. This illustrates how the Unify framework enables DevOps related mechanisms, with automated monitoring and debugging features. Figure 20 depicts this integrated prototype. The architecture is divided over 3 main nodes:

- The **Global Orchestrator Node** is primarily running ESCAPE (cf. the service layer and (global) orchestration layer in Figure 18). This is where the initial NF-FG is generated.
- The **Universal Node** deploys any received NF-FG locally (cf. the infrastructure layer in Figure 18). It also includes a monitoring controller (in the UN example called MMP) that instantiates and configures monitoring related functions according to the definitions in the MEASURE annotations of the NF-FG.
- The **Troubleshooting Node** executes the semi-automatic debugging and troubleshooting tasks, triggered by the monitoring framework.

A short explanation is given of every functional block in this diagram. More details can be found in the respective deliverables or references.

Name	Description	Info
ESCAPE	Combines the service layer and global orchestrator functions in this prototype.	Documented in [D3.4].
DDBroker	DoubleDecker, a scalable messaging bus based on ZeroMQ that features built-in message filtering and aggregation and sending opaque data between virtual network and monitoring functions, as well as towards higher layer orchestration modules.	Documented in [D4.3] and [D4.4].

Table 5: Functional blocks of the global orchestrator node.

Name	Description	Info
UN orchestrator	Local orchestrator of the UN.	Documented in [D5.5]
Docker deamon	Deamon capabale of starting VNFs packaged as Docker containers.	Open-source software [Docker]
Monitoring Management Plugin (MMP)	This entity represents a monitoring controller inside a UN, and deploys and configures the monitoring tools and functions requried to realize monitoring metrics as described in the MEASURE annotations.	Documented in [D4.3] and [D4.4].
DDBroker	Locally deployed DoubleDecker message bus on the UN. The alarms generated by the monitoring tools are published via this message bus.	Documented in [D4.3] and [D4.4].

Table 6: The different functional blocks of the UN (related to the elastic router use-case).

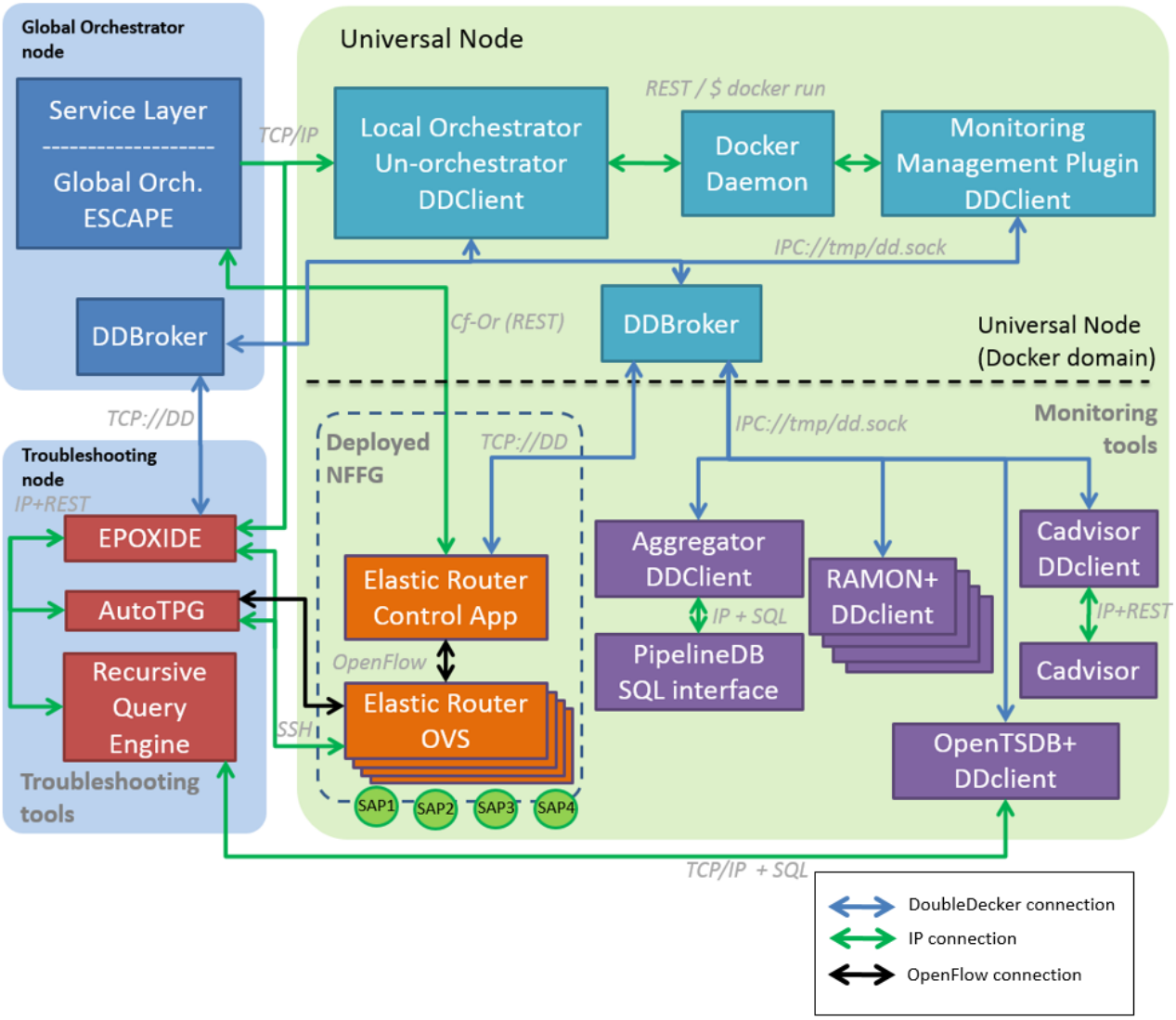


Figure 20: Integrated prototype with all functional blocks and their interfaces, demonstrating the elastic router use-case.

Name	Description	Info
Control NF	OpenFlow controller (Ryu) with an extra layer which handles the NF-FG updates. It has a external interface that connects to the docker0 bridge (not to the LSI). This is used to create the cf-or interface and connection to the DoubleDecker message bus	Functionality described in [D3.3]
OVS VNF	OpenVswitch instance that routes the traffic between the connected SAPs. It has a control interface that connects to the Control NF via the LSI. It also has a public interface, which is connected to the docker0 bridge (not to the LSI) and is reachable from outside. This interface exposes an SSH interface to which eg. troubleshooting tools can connect.	Functionality described in [D3.3]

Table 7: The different VNFs deployed as Docker containers, as part of the elastic router service.

Name	Description	Info
RAMON	A tool for determining a statistical estimate of link utilization and that offers congestion indications. One RAMON instance per monitored port needs to be deployed.	Documented in [D4.3] and [D4.4].
Cadvisor	Provides resource usage and performance characteristics of the running VNF Docker containers.	Open-source software [Cadvisor]
Aggregator	Aggregation point, used to aggregate raw metrics streamed from monitoring functions such as RAMON or CAdvisor via DoubleDecker. In this use-case, metrics from the monitoring functions are collected and assessed for the purpose of triggering scaling when needed relative to defined scaling conditions.	Documented in [D4.3] and [D4.4].
PipelineDB SQL interface	An open-source relational database that runs SQL queries continuously on streams, incrementally storing results in tables. Supporting function to realize the aggregation point in this use-case	Open-source software [PipelineDB]
OpenTSDB	A scalable, distributed time series database that logs the monitored metrics from RAMON and Cadvisor.	Open-source software [OpenTSDB]

Table 8: The different monitoring related components deployed as part of the elastic router service.

Name	Description	Info
EPOXIDE	Emacs-based framework capable of assembling general and SDN specific debugging and troubleshooting tools in a single platform and make it easy for a developer to combine them.	Documented in [D4.3] and [D4.4].
AutoTPG	Tool to verify the flow-matching functionality of OpenFlow switches/routers.	Documented in [D4.3] and [D4.4].
Recursive Query Engine	The recursive query language (RQL) is used to query monitoring metrics on high level of abstraction, corresponding to NF-FG and SG abstractions in higher layers of the UNIFY architecture.	Documented in [D4.3] and [D4.4].

Table 9: The different tools deployed in the troubleshooting node.

4.1.4.1 Deployment and monitoring message chart

Using the integrated prototype from Figure 20, a workflow can be installed to deploy and monitor the elastic router NF-FG. The message chart in Figure 21 shows the deployment procedure as described in Section 4.1.1 (step 1-4). Once the VNFs in the elastic router NF-FG are deployed, the UN local Orchestrator triggers the Monitoring Management Plugin to start the monitoring actions defined in the MEASURE annotations (step 5). The monitoring framework configures the monitoring components and analyzes the captured metrics (step 6-9). Once the monitoring tools detect an alarm (a monitored metric has surpassed a threshold defined in MEASURE), this alarm is published on the DoubleDecker message bus and received by the Control NF (step 10). This triggers a scaling action and the Control NF sends an updated NF-FG to the global orchestrator (step 11). More details of this monitoring support for the elastic router use-case is described in [D4.3] (Section 6).

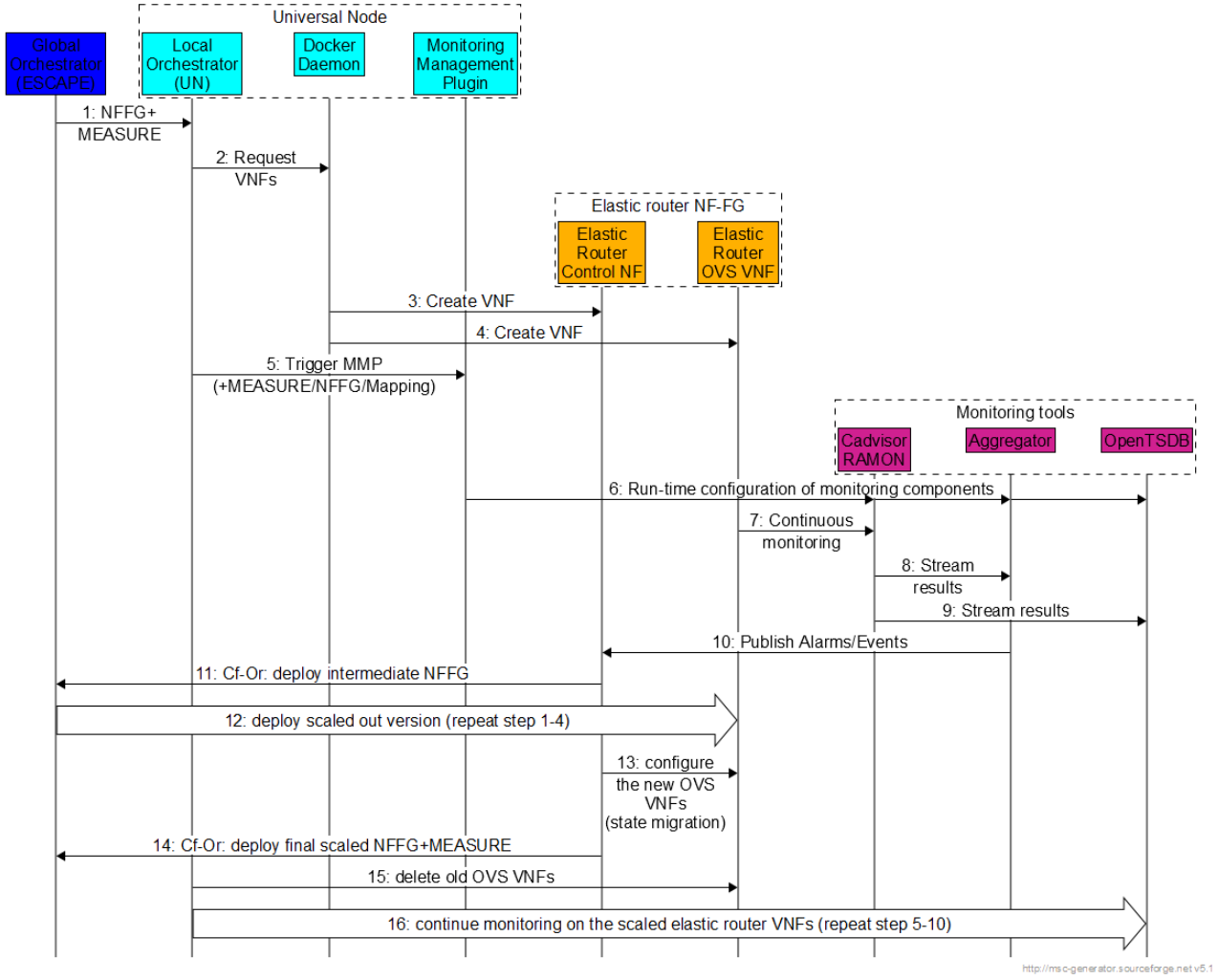


Figure 21: Deployment and monitoring message chart.

4.1.4.2 Troubleshooting message chart

In addition to deploying and monitoring the elastic router service, an automated troubleshooting workflow can be installed in the prototype of Figure 20. The message chart in Figure 22 shows the troubleshooting procedure. The process is triggered by monitoring certain metrics (step 1-3) which are gathered and analyzed in the Aggregator. The Aggregator will then trigger EPOXIDE in case of a detected anomaly (step 4). In the next steps, EPOXIDE will trigger the Recursive Query Engine to query the monitored data, in order to make a decision which VNF needs further troubleshooting. In case an OVS VNF is estimated to be malfunctioning, AutoTPG will be started to investigate in an automated way the forwarding behaviour of the chosen OVS VNF. The detailed scenario of the troubleshooting example is further described in [D4.3] (Section 6).

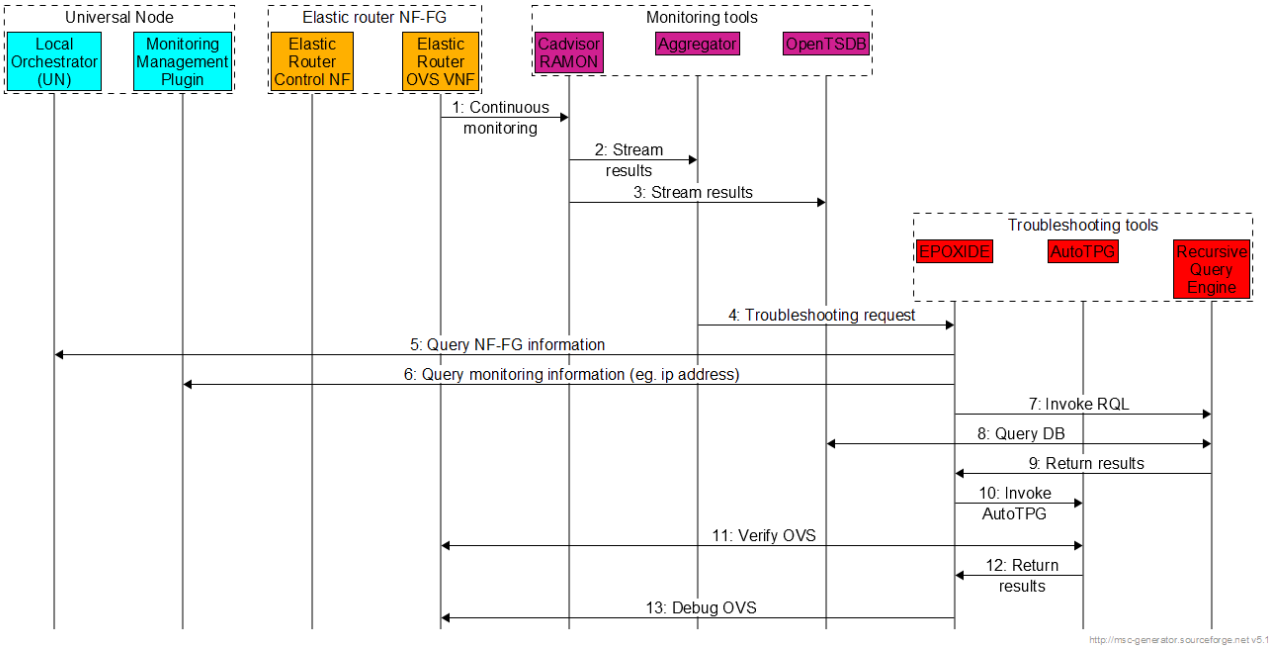


Figure 22: Troubleshooting message chart.

4.2 Integration of FlowNAC and state migration (FlowSNAC)

FlowNAC, introduced in section 3.3.2 of [D3.2], is a Flow-based Network Access Control (NAC) solution that allows to grant users the rights to access the network depending on the target service requested. Each service, defined univocally as a set of flows, can be independently requested and multiple services can be authorized simultaneously.

Traditional approaches to NAC solutions implement them as monolithic software packages. All the traffic from users must be redirected to the NAC element by the network infrastructure. The Authentication and Authorization (AA) traffic is processed, authorized data traffic is allowed back into the network to reach the service, and non-authorized data traffic is dropped.

FlowNAC exploits the programmability of Software Defined Networking (SDN) datapath elements to perform the stateless processing of the data traffic, and decomposes the functionality of the NAC solution in several micro-services, shown in Figure 23 and described next.

FlowNAC Control App (FNC)

A stateful component that keeps the state of the AA processes currently executed, for one or more FlowNAC Enforcing Blocks (FNEs) and the end users attached to them. The FNCs receive the AA traffic (Extensible Au-

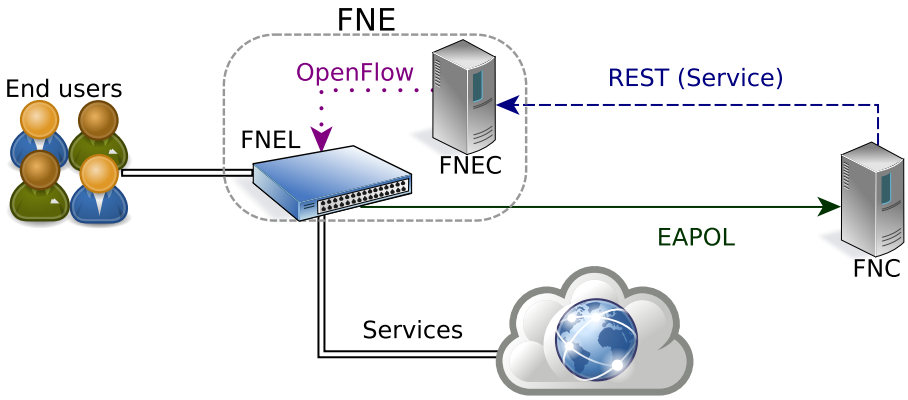


Figure 23: The different microservices composing the FlowNAC service

thentication Protocol over LAN (EAPoL)) from the end users and, upon successful authentication, open the corresponding service paths in the FNE. Each FNC periodically notifies the FNS of the current load.

FlowNAC Enforcing Block (FNE)

A stateless component responsible for limiting the access to the network only to previously authorized services. When the end users request access to different services, the requests are forwarded to the FNC which, upon successful authentication, configures the FNE to allow the user reaching the requested services. The FNE is composed by two subcomponents, the **FlowNAC Enforcing LSI (FNEL)**, that will be deployed using network resources and implements the enforcing, and the **FlowNAC Enforcing Controller (FNEC)**, that will be deployed using compute resources, controls the FNEL and provides a REST interface towards the FNCs. The main reasons for this split are discussed in section 4.2.5.4

This decomposition allows redirecting each type of traffic to the appropriate resource. Only AA traffic is sent to the VM to be processed, which configures the SDN element to allow authorized services. Data traffic is processed by the SDN switch: only authorized traffic is allowed and unauthorized traffic is dropped.

This approach fosters the scalability of the solution, as each block has different requirements and uses different resources. Performance is also improved, as specialized hardware can be used to process the most intensive and bandwidth consuming traffic. The service placement process and overall resource utilization also benefit from the decomposition. In FlowNAC, for example, the decomposition enables a twofold bandwidth use reduction:

- Separating the components allows a wider placement scope, for example the FNE can be deployed in the first element in the datapath and thus prevent unauthorized traffic from consuming network resources;
- Redirecting only the AA traffic to the FNC streamlines the authorized data traffic path, as it does not have to be processed by computing resources. The network devices are the basic substrate for building the service chains, which means that they must be crossed anyway and their availability as a resource is more widespread. On the contrary, the commodity servers can be limited to specific locations, for example data centre infrastructures, thus requiring a more complicated traffic steering and, possibly, imposing some delay due to the longer path the traffic must traverse.

4.2.1 Motivation

Here we wish to demonstrate two main aspects, and the extensions needed to implement them. Firstly, we wish to highlight the possibilities provided by the Cf-Or interface, which allows a running service to dynamically modify its

resource allocations through modification of its own NF-FG. This mechanism provides the basic functionality needed to react to shifting demands on the service, as well as to react to failures of components that are part of the service.

Secondly, we want to show the use of features provided by SecOpenNF to migrate state in a security-related VNF system as a mechanism for providing load balancing and resiliency without service degradation. While the Cf-Or interface allows the addition or removal of resources, simply adding/removing resources to/from the system will in many cases cause services to fail or degrade when the involved components are stateful.

4.2.2 Related work

An introduction to the issues surrounding state migration for handling service scaling was given in Deliverable 3.1, section 6.7.2 [D3.1]. Deliverable 3.2, section 5.2, and the publication [KDS15] describes the extension of an existing mechanism for handling state, OpenNF [Gem+14], to better meet carrier-grade performance requirements. These extensions allows the bulk of the state migration process to be handled in a distributed fashion, controlled from a central point, without losing the guarantees provided by the previously fully centralized system. In D3.2 the main use-case scenario considered was service scaling through scale-out or -in, i.e. by adding or removing VNFs to a service[D3.2]. Deliverable D3.3, section 2.6, describes a load-balancing and a resiliency scenario applying the extended OpenNF mechanism to a concrete service scenario, FlowNAC [D3.3].

FlowNAC [Mat+14] differs from the VNF studied in earlier deliverables by being what we call a "Split State" VNF, i.e. state is distributed among several VNF components, performing different roles and split following the proposal in [Mat+15]. In previous deliverables the VNFs had been "monolithic", i.e. all required state was maintained within a single VNF. FlowNAC also differs from the previously studied VNFs as it performs a sensitive security role within the network, authenticating customer connections and authorizing services.

Here we build upon the work described in these previous deliverables to evaluate if the extended OpenNF mechanisms are adequate as a means to implement resiliency and load-balancing for collections of "Split-State" VNFs, such as the FlowNAC case. To increase the security of the system we utilize the DoubleDecker messaging system, developed in WP4, to provide a secure transport layer with point-to-point and publish/subscribe capabilities. The DoubleDecker messaging system has been described in Deliverable 4.2, section 5.1, and publication [D4.2; Joh+15].

4.2.3 FlowSNAC - Extending FlowNAC for load balancing and resiliency

The design of FlowNAC, split into a stateless enforcing block (FNE) and a stateful control app (FNC) can be leveraged to efficiently provide load balancing and resiliency capabilities. Since all the transient state is kept in the FNC, only these components require more complex and state-aware processes, as supported by SecOpenNF, whereas simpler methods can be used for the FNEs.

Also, the lack of state in FNE means we can easily add new FNE blocks if needed and configure then from the FNC. Migrating the FNEs between different FNC does not impact the already deployed services while the migration is in process, since the FNE is already configured to allow these services through. In section 4.2.5 we discuss the overall impact of the proposed solutions to the FlowNAC service.

To support dynamic load balancing and resiliency in FlowNAC we have extended several of the FlowNAC components and added additional control and signalling components. The FlowNAC components have been updated so they can be dynamically instantiated by the Resource Orchestrator and the complete service can be reconfigured, allowing us to react to failures and varying load by allocating additional components, and to avoid overprovisioning by deallocating under-utilized components. All the state transfer processes use as a grouping criteria the FNE the users are connected

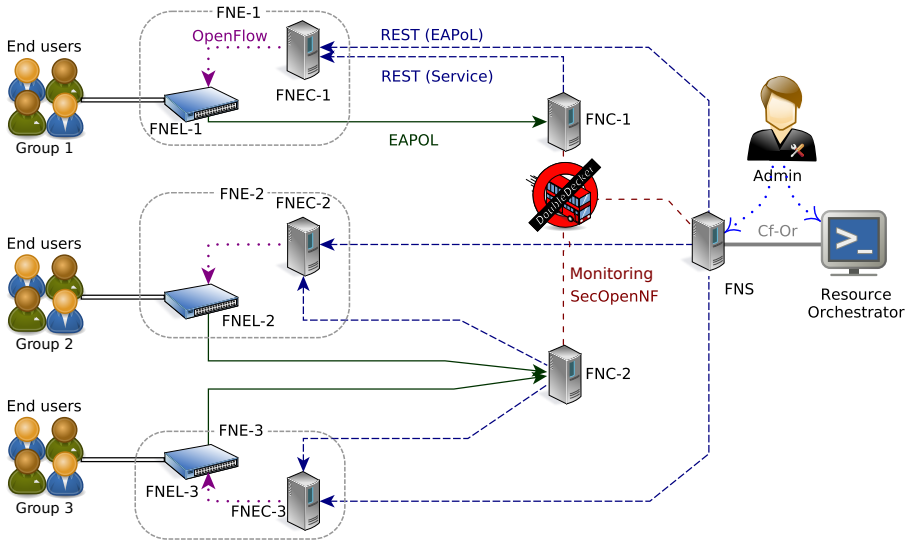


Figure 24: The extension of FlowNAC to FlowSNAC adds three additional components, the Resource Orchestrator, the FNS, and the DoubleDecker broker. In additon, several existing components have been extended.

to, so all the users connected to FNE x are referred to as Group x.The resulting system can be seen in Figure 24. The components that have been extended are:

FlowNAC Control App (FNC)

Has been extended to support several operations on the transient state, this includes support for inserting, exporting, updating, and deleting transient state. The FNC has also been extended to support the SecOpenNF, which is the combination of the distributed OpenNF developed earlier (see [Gem+14; KDS15]) with DoubleDecker for secure transfer of state. SecOpenNF allows secure, distributed, transfer of state. Finally, the FNC has been extended to act as a Monitoring Function, but sending measurements of its own liveness and internal load.

FlowNAC Enforcing Controller (FNEC)

Has been extended with policy control to restrict which commands the FNS and the FNCs are allowed to execute. This is to ensure that the FNS is only allowed to modify the EAPoL flowspace, in order to redirect the authentication traffic to the assigned FNC, and to ensure that FNCs are only allowed to modify the flowspaces of deployed services and cannot affect the EAPoL traffic.

In addition to extending these components, several new components have been introduced:

Resource Orchestrator

A central component of the UNIFY framework responsible for deploying resources requested from the Service Layer. It also deploys additional resources requested by the FNS through the Cf-Or interface.

FlowNAC Service Controller (FNS)

Is responsible for coordinating many actions in the system. It coordinates the transfer of state between FNCs, by either commanding an FNC to continuously synchronize state with another FNC (for resiliency), or by commanding an FNC to transfer the state of a group of users to another FNC (for load balancing). The FNS also is responsible to coordinate the network reconfiguration to direct EAPoL traffic from each FNE to the appropriate FNC, done through a REST API exposed by the FNEs. The FNS also communicates with the Resource Orchestrator through the Cf-Or to allocate and de-allocate resources, in order to instantiate additional nodes for scaling

and/or for backup purposes. The FNS is also the recipient of monitoring information from the FNCs which it uses to monitor their load and liveness. Finally, it exposes a REST API that can be used by and administrator to monitor the status of the overall FlowSNAC system, and to trigger administrative actions such as enabling a backup FNC. It is acting as a Scalability management ControlApp mentioned in section 3.4.1 in [D2.1].

FNS CLI Client

To perform administrative actions using the FNS REST API we have implemented a CLI interface which communicates with the FNS. Using this CLI one can monitor the load of the FNCs, configure FNC resiliency, as well as monitor the status for load balancing and resiliency processes.

DoubleDecker broker (DD)

The DoubleDecker broker is a simple distributed messaging system with security features, based on ZeroMQ. It provides connectivity for clients over either TCP or IPC mechanisms using a simple protocol, together with mutual authentication and end-to-end encryption based on Elliptic Curve Cryptography. It supports two messaging patterns, point-to-point notification between authenticated clients, and a publish/subscribe mechanism. It is also multi-tenant with strict isolation and supports dynamic topological restrictions on publish/subscribe topics. It is responsible for securely transferring SecOpenNF commands between FNS and FNCs, and securely transferring state between FNCs. It is also used as the communication channel between the FNS and the Resource Orchestrator, for the Cf-Or interface.

Secure Distributed OpenNF (SecOpenNF)

An extension of the OpenNF middleware, which integrates with VNFs to provide reliable, consistent state migration[Gem+14]. To provide reliability it synchronizes the migration of VNF internal state with required network updates, ensuring that packet are not lost nor re-ordered during the process. In previous work we described how OpenNF was extended to support distributed (peer-to-peer) transfer in order to improve the scalability of the migration process [D3.2; KDS15]. To provide a scalable and secure communication channel for the sensitive AA state we have extended OpenNF further by using the DoubleDecker messaging system [D4.2], for both control and state transfer messages. It is integrated with the FNCs and the FNS.

4.2.4 Load balancing and resiliency processes

A high-level view of the Resiliency and Load Balancing processes are show in Figure 25, where different stages of the service lifecycle are shown. A high level message sequence diagram going between these stages is shown in Figure 26, note that all communication between FNS and Resource Orchestrator is through the Cf-Or using an NF-FG. Also note that the different users connected behind each FNE are assigned to groups (Group 1 and 2), this assignment is based on the users MAC addresses. All users are assigned a MAC address, with users in the same group sharing a MAC prefix. This is used to identify all the traffic flows for the individual users and group them. Going through the stages in the processes one by one:

Stage 1 In the initial deployment an FNS, an FNC (FNC_1), and a DoubleDecker broker are deployed and interconnected. Once the FNS is running, it requests the NF-FG it is part of from the Resource Orchestrator, in order to discover the other components. The deployed FNS connects with FNC_1 to configure the associated groups and the corresponding FNE REST API. The FNS uses that REST API to configure FNE_1 and FNE_2, which are running at customer premises, to direct EAPoL traffic to FNC_1. Once this has been done Users can authenticate with FNC_1, which in turn can start deploying network services to authenticated users.

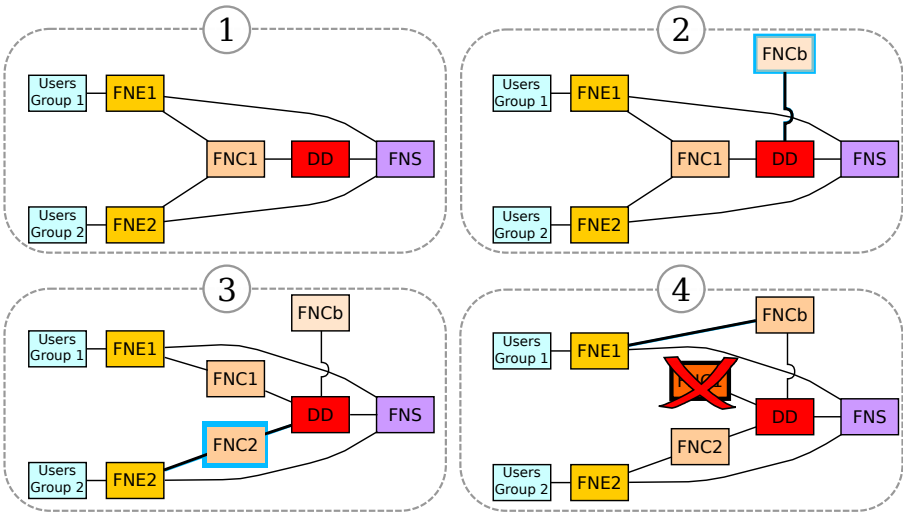


Figure 25: The deployed service after the different stages, not showing the FlowNAC steps of authenticating and deploying services to users. Stage 1) represents the initial deployment, 2) the addition of a backup FNC, 3) scale-out due to high load where FNC_2 is added, and 4) trigger of failover from FNC_1 to FNC_b due to a failure.

Stage 2 The deployment of a backup FNC (FNC_b) is triggered by an administrator. The administrator does this by issuing a command to the FNS REST API, which in turn causes the FNS to instruct the Resource Orchestrator to allocate an additional FNC for backup, using the Cf-Or interface. Once the backup FNC is running, the FNS triggers the start of state synchronization from FNC_1 to FNC_b using SecOpenNF.

Stage 3 Through the load monitoring information the FNS has received from FNC_1, the FNS observes that the load on FNC_1 has exceeded a predefined threshold and triggers the scale-out procedure. The scale-out procedure uses the Cf-Or interface to allocate an additional FNC (FNC_2) and connect it to the user-groups that are to be balanced. After deployment, and once connectivity has been established, the FNS triggers the migration of state from FNC_1 to FNC_2. Once state for a group of users have been transferred, the FNS updates the group association in each FNC and redirects the EAPoL traffic for that user group from FNC_1 to FNC_2.

Stage 4 Through liveness monitoring the FNS observes that FNC_1 has had a failure and triggers the redirection of EAPoL traffic from FNE_1 to FNC_b, at the same time FNC_b is informed by the FNS that it is responsible for enabling services at FNE_1 and how it can reach the REST API of FNE_1.

The next sections will go into further details, with a closer description of the FNC monitoring, the load balancing, and resiliency processes. These processes are fully managed by the FNS, which holds the service specific control logic.

4.2.4.1 FNC Monitoring

FNCs are both processing incoming network packets, as well as transmitting monitoring data concerning their internal state (number of connected users and liveness), and are therefore acting both as VNFs as well as MFs. The monitoring data is published via the DoubleDecker broker to the FNS, the publication of these messages starts once a FNC has registered with the FNS. The FNS in turn uses this information for two purposes, to 1) verify the liveness of the registered FNCs, and to 2) estimate the load on FNCs based on the number of End users connected.

4.2.4.2 Load Balancing Process

The load balancing process is triggered by the monitoring information sent by the FNCs, which is used by the FNS to estimate the load on the FNCs and implement a policy. According to this policy, if the number of users on one FNC exceeds a predefined threshold a scale out process is triggered, starting another FNC and

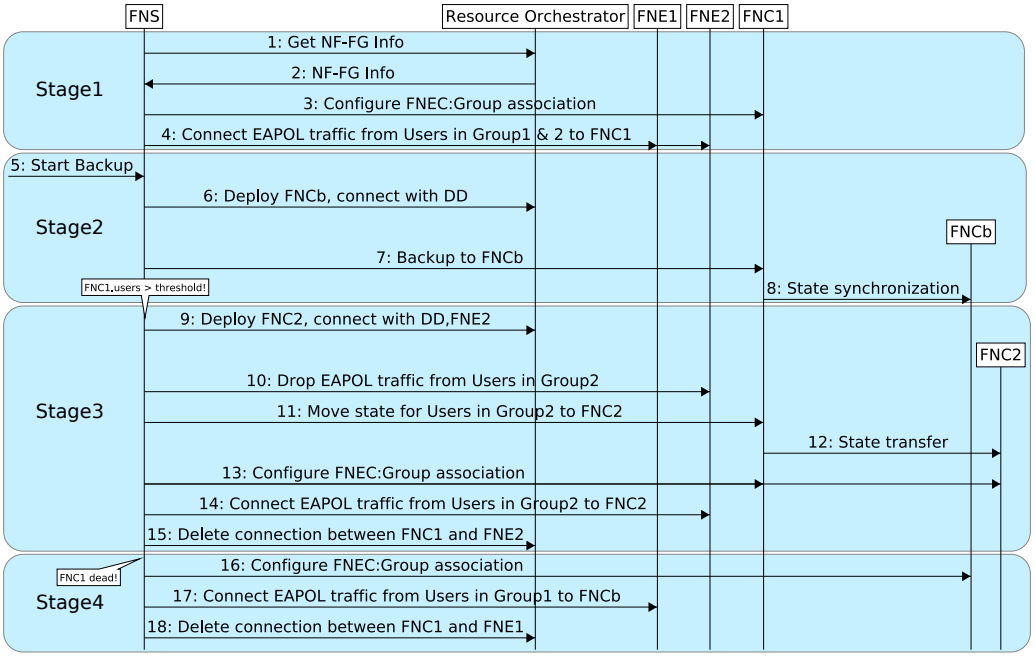


Figure 26: Message sequence diagram showing the high level interactions between components when going through the four stages in the life-cycle depicted in figure 25.

transferring half of the users from the exceeding FNC to the new one. Conversely, if the number of users of more than one FNC are below a threshold, a scale in process is triggered. In the scale in process, users from one FNC are migrated to the other FNC, and the empty FNC is de-allocated.

When the policy indicates a scale-out, the FNS determines which user groups should be transferred and modifies the NF-FG it has retrieved from the Resource Orchestrator. It starts by making a copy of the overloaded FNC (FNC_{src}) in the graph, creating a new FNC (FNC_{dst}) node with links to the components of the FNEs for the groups to be migrated (FNELs and FNECs) and links to the DoubleDecker broker and the FNS. This modified NF-FG is then sent to the Resource Orchestrator for deployment.

Once the modification has been deployed the FNS obtains the latest version of the NF-FG and determines the DoubleDecker identifier of the newly deployed FNC. The service is now in an intermediate stage, where both FNC_{src} and FNC_{dst} have connectivity to one or more of the same FNEs simultaneously.

Based on the information the FNS has obtained, it starts moving state. It does this by sending a MoveMultiflow message to FNC_{src} containing a key indicating which state should be moved, this key is the MAC prefix identifying the user group, and the source and destination of the state transfer (FNC_{src} and FNC_{dst} respectively). Upon receiving this message FNC_{src} gathers all its internal state matching the key into a list of state chunks. These chunks are then sent to FNC_{dst} one by one in MovePerFlow messages. These additionally contain a transaction ID and each chunk's key. When FNC_{dst} receives a MovePerFlow it inserts the state into its internal data structures and acknowledges the reception with FNC_{src} which then removes the transferred state from its own structures.

When all state identified by the key has been transferred to FNC_{dst}, and deleted from FNC_{src}, FNC_{src} informs the FNS that the MoveMultiflow operation has been completed. The FNS then contacts the FNEC responsible for the users who's state has been transferred and tells it to redirect future EAPoL traffic to FNC_{dst} rather than FNC_{src}, this is done using the REST API exposed by the FNEC. Finally, the FNS once again modifies the NF-FG. This time the FNS removes the links between FNC_{src} and FNEs that now are managed by FNC_{dst}. Once the updated NF-FG has been sent to the Resource Orchestrator and deployed, the scale-out process is completed.

In the scale in process, once the trigger is met the FNS decides which FNC must be decommissioned (FNC_{src}) and which FNC will take over the groups to be migrated (FNC_{dst}). The FNS requests to the Resource Orchestrator to add the corresponding links. Then instructs the FNC_{src} to transfer all the state to FNC_{dst}, updates the FNECs for the migrated groups and modifies the NF-FG to remove the migrated FNC and the links.

4.2.4.3 Resiliency process The resiliency process starts when then FNS first retrieves the initial NF-FG from the Resource Orchestrator, when the FNS was just deployed. The FNS then scans the NF-FG for FNCs marked for resiliency and implements the resiliency policy. According to this policy, a backup FNC is added to NF-FG (N+1 redundancy), connected only to the DoubleDecker broker. The modified NF-FG is then sent to the Resource Orchestrator for deployment.

Once deployed, the FNS determines the DoubleDecker ID of the newly deployed FNC, e.g. FNC_b. The FNS then signals all previously running FNCs to start synchronizing state, using the ActivateBackup message which contains the FNC_b DoubleDecker ID. FNCs that receive the ActivateBackup message first duplicate their existing state to FNC_b, and acknowledge the operation to the FNS. Any future updates to the state in a FNC will be automatically synchronized with FNC_b, either to remove, update, or insert new state.

With the backup FNC running and synchronized, the FNS can react to loss of liveness of any other running FNCs. When it detects that a FNC is dead the FNS scans the NF-FG for FNC_b and copies all the links from the dead FNC to the FNEs the dead FNC was responsible for to FNC_b. The FNS then sends the modified NF-FG to the Resource Orchestrator for deployment. This step could be avoided if these links are pre-provisioned, in a trade-off between the resources committed and the reconfiguration delay added to the resiliency process.

Once deployed, the FNS calls the REST API of the affected FNEs and redirects the EAPoL traffic to FNC_b, which now is responsible for the affected user groups. Once all EAPoL traffic has been transferred the FNS once again modifies the NF-FG, this time it removes all links from the dead FNC and marking it as broken. Once this NF-FG has been deployed the failure has been handled, and a troubleshooting process can be initiated to investigate what caused the failure. The FNS the re-initiates the process to recover the defined resiliency model.

4.2.5 Discussion

4.2.5.1 Impact on the AA process In order to properly analyze the impact of the load balancing and resiliency scenarios in the AA processes we must distinguish between three cases:

1. AA processes already completed and which have granted the users access to the requested services.
2. AA processes that start while the scaling or backup processes are being carried out.
3. AA processes that are in an intermediate state when the scaling or backup processes are being carried out.

Due to the split design of FlowNAC, the completed AA processes are not impacted. The FNEs have been configured to allow the authorized services so, even if the FNC is nos available for a period of time, the FNE will continue to allow them through. Connection of the new FNC doesn't impact those services either as there is no need for the new FNC to update them.

For the AA processes starting during the state migration process, the key criteria is the duration of the state migration process compared to the timeouts defined in the AA process. When the FlowNAC client sends a request to the FNC, it will wait for a defined time for the answer before sending a new request. After a defined number of retries, the FlowNAC client will report an error in establishing the connection. If the state migration process is completed before the retries have been exhausted, the AA process will be successfully completed without further intervention, even if

with a delay. If the state migration process takes longer, the AA process will have to be restarted, either manually or by whichever process was launching the FlowNAC client.

As these AA processes have no associated state, one possible improvement to this situation could be to redirect the new petitions from the FNEs to be migrated to the new FNC from the moment the new FNC is available, instead of waiting for the state migration process to be completed. Due to the complexity added to the state migration process, this alternative would only be required if the performance of the state migration process would demand the FlowNAC timeouts to be set to a unacceptably high value for the deployed scenario.

For the AA processes in an intermediate state, again the key criteria is the duration of the state migration process compared to the timeouts defined in the AA process. Besides, there is the possibility in the resiliency scenario that some state is lost (changes to the state made since the last state synchronization until the monitoring detects that the FNC is down). The timeouts built in the FlowNAC client help again gracefully recovering from these situations, as long as the state migration process completes before the FlowNAC client decides that the connection is not successful.

To characterize these scenarios, we will carry out performance measures of the state migration process to provide a guidance for the timeout values to be considered depending on the expected load of the FNCs.

4.2.5.2 Monitoring In the observability process, monitoring requirements for a service should be described using the MEASURE language, that is used by an Monitoring Controller (MC) or Monitoring Management Plugin (MMP) to start, stop, and configure monitoring functions (MF) able to monitor the requested parameters, and configure an aggregation point for digesting results. The aggregation point receives monitoring results from MFs, applies the aggregation logic described in the MEASURE language, and transmit alarms to concerned components whenever e.g. defined thresholds have been exceeded. The monitoring implemented in this demo follows the concepts defined in the WP4 Observability processes, and could use the MEASURE language (see section 3.2, section 3.3, and section 5.2 in D4.2 [D4.2]). However, due to prioritzing the state migration processes rather than observability, the full observability procsses have not been integrated (these are on the other hand shown in the Elastic Router demo, see section 4.1).

While the general process defined in WP4 is still valid for the use-cases implemented in this demo, we discovered some deficiencies in the MEASURE language that could be remedied to better support similar scenarios. First, the MEASURE language assumes that only MFs generate monitoring results, these monitoring functions are under control by the MC/MMP. Secondly, the MEASURE language assumes that actions are taken as a reaction to monitoring results, consequently it lacks a mechanism for specifying reactions based on not receiving data.

In this demo scenario monitoring results, the number of users managed by a FNC, is generated not by an MF but by a VNF, the instantiated FNCs. Monitoring result variables in MEASURE are defined as $\mathrm{variable} = \mathrm{metric}(\mathrm{parameters})$, for example $\mathrm{m1} = \mathrm{delay}(\mathrm{from{=}port1,to{=}port2})$. When processed this triggers the start and/or configuration of an MF able to provide the metric, for example an MF performing ping, this needs to be extended to support the case of VNFs generating results. One solution could be to define a special metric keyword that refers to a VNF in the NF-FG associated with the MEASURE description, together with a description of the generated monitoring result. This could for example be $\mathrm{m1} = \mathrm{fromVNF}(\mathrm{FNC}_1, \{\text{'users':'int'}\})$ showing that FNC_1 will generate results containing the integer metric, 'users'. This information would be enough for the MC/MMP to resolve the name of the instanced FNC_1 and configure the aggregation point. No other change to MEASURE would be needed to support VNFs that generate monitoring results.

This demo also relies on heartbeats to indicate that a service is alive, reactions are triggered when no heartbeats are received within a certain time. MEASURE on the other hand assumes that reactions should be triggered based on the data arriving in monitoring results. One way remedy this would be way to express temporal aspects as part of MEASURE's zone logic which describes when a metric has moved into a separate zone, which in turn can have reactions

associated with it, and logic in the Aggregation point to maintain message arrival times. The MEASURE description of this solution could be as shown in Figure 27, to the right.

```
measurements {
 m1 = liveness(from=FNC_1, param =
 {'users':'int'}, timeout=30s);
} zones {
 z1 = m1 > 0;
 z2 = m1 < 1;
} action {
 z1->z2 = Publish(alarm,..);
}
```

```
measurements {
 m1 = fromVNF(FNC_1, {'users':'int'});
} zones {
 z1 = m1.age > 30s;
 z2 = m1.age < 30s;
} action {
 z2 -> z1 = Publish(alarm,..);
}
```

Figure 27: Liveness solution with an external liveness MF (left), and with an internal age parameter (right).

Another solution could be to place this logic in an special MF that can be configured to receive messages from a VNF and transmit a 'monitoring result' when messages have not be received for a certain time. This solution requires no modification to the MEASURE language and model, only a new MF. Assuming that 'liveness' is a metric that is 1 when a VNF is alive and 0 when not alive, this solution in MEASURE could be as shown in Figure 27, to the left. For this demo these solutions have not been implemented, as the purpose of the demo is not to highlight MEASURE and the observability process, rather we see these findings as feedback to WP4.

4.2.5.3 DoubleDecker broker deployment When deploying the DoubleDecker broker there are three basic deployment options:

1. Infrastructure mode. The broker is deployed as part of the bootstrapping of a Universal Node, any deployed VNF can authenticate with it and take advantage of its services in order to communicate with other VNFs or components. By adding additional keys to the running broker, new tenants can be accommodated.

2. Shared mode. The broker is instantiated as a VNF in an NF-FG, and shared between multiple NF-FGs that explicitly connect VNFs that should use the service to the DoubleDecker broker VNF. By adding additional keys to the running broker, new tenants can be accommodated, however this has to be done by the owner of the broker VNF who likely does not want to share it with other tenants.

3. Single-service mode. A broker is instantiated as a VNF in a single NF-FG, with explicit links to VNFs that should be able to connect to it. Sharing between NF-FGs is not allowed.

In the FlowSNAC setup any of these options is feasible, with some consequences. It is worth noting that we likely have two different tenants in the system, that should be isolated. The first is the owner of the service itself, who is buying resources that are managed by the Resource Orchestrator, and the owner of the infrastructure, Resource Orchestrator etc. To ensure that traffic between parts of FlowSNAC is not monitored by the infrastructure owner, these should be encrypted and readable only by the FlowSNAC owner. The two need to be able to communicate however, in order to realize the Cf-Or interface.

Here it makes sense for the infrastructure owner to run as the "public"[7] DoubleDecker tenant, which is a special super-user tenant able to communicate with clients of other tenants, in a secure manner. Normally, different DoubleDecker tenants are completely isolated from each other and cannot communicate in any way, the special "public"

[7]The name "public" refers to the concept of public services, provided by a "super-user" in society, e.g. the government, for everyone to use. This does not imply that clients in the public tenant are completely open, free, etc.

tenant is the only exception to this. Running certain special services in the public tenants, such as the Resource Orchestrator, allows Control Applications running as part of different tenants to communicate their needs to the Resource Orchestrator, without risking leaking any other information to either each other or the infrastructure owner.

For the "public" tenant, an infrastructure broker is the simple solution, as it is available from the bootstrapping and any tentant client can use it. For the FlowSNAC internal communication an infrastructure broker makes equal sense, as long as it is providing similar performance as a shared/single mode broker. If the infrastructure broker is heavily loaded, it makes sense to deploy a private broker for internal messaging.

4.2.5.4 FlowNAC Enforcing Block (FNE) component separation In FlowNAC the FNE is split into two subcomponents: the FNEL and the FNEC. Since the main role of the FNEC is to provide a REST interface for the FNEL towards the FNC, it could be argued that a single FNEC, collocated with the FNC would be sufficient and demand less resources.

There are three main reasons for the proposed approach:

1. The connection between the FNE and FNC must be reconfigured in the load balancing and resiliency processes. If the FNEC is collocated with the FNC, the reconfiguration requires establishing a new switch-controller association. Locating the FNEC with the FNEL allows having a stateless interface (like REST) instead, making significantly easier the FNE-FNC connectivity.

2. The connection to the FNE is shared between the assigned FNC and the FNS, and each of them is only allowed to perform a restricted set of operations. The REST interface provided by the FNEC again simplifies the implementation. The alternative requires adding also an FNEC to the FNS and handling a multiple controller scenario.

3. When the FNE is deployed in a CPE, a local control function enables the deployment of local services, provided without involving resources located in the operator network and, as a consequence, resilient to losses of connectivity.

4.3 Integration of decomposition with the embedding algorithm

In order to support service decomposition, the Network Function Information Base (NF-IB) should be able to store the NF models/abstractions, NF relationships, NF implementation image(s) and NF resource requirements. We have implemented the NF-IB in Neo4j database. As this database is capable of storing key-value pairs for nodes and edges, for each NF we have stored the tree-like structure (supporting the decomposition process) with all the corresponding information of nodes and edges. This was explained in detail in [D3.3].

In order to jointly decompose services and allocate resources to the NFs, the embedding (mapping) module should be able to choose a suitable decomposition for the NFs of the requested service (i.e., retrieve the NF decompositions from the NF-IB and find a suitable selection) and find a close to optimal mapping for the elementary NFs. To this end, we have integrated a heuristic-based embedding algorithm which supports service decomposition to the ESCAPE framework. This algorithm was explained in detail in [D3.2].

There are three main steps within the implemented mapping strategy:

1. Retrieve all decompositions for the requested NFFG from the Neo4j-based NF-IB

2. Select a minimum cost decomposition. For each decomposition we calculate a cost based on:

    - Number of potential physical nodes which can host the NFs of the decomposition

- Number of NFs in the decomposition
- Cluster Factor (CF) of the decomposition (how same-type NFs are connected, Fig. 28 indicates two decompositions with CF=2 and CF=4 respectively.)

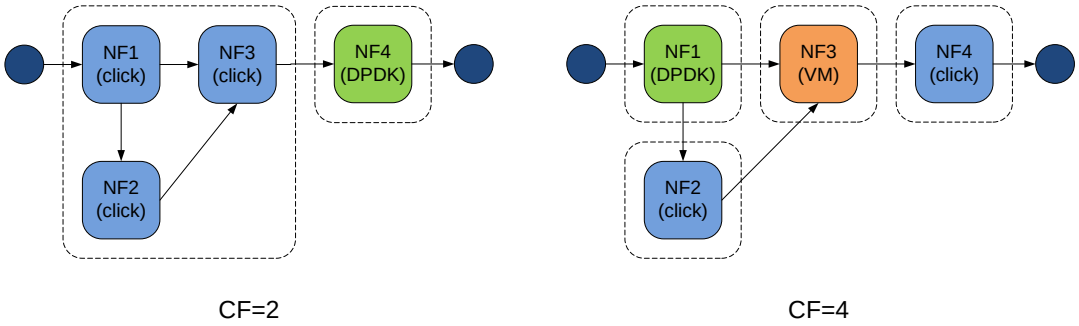


Figure 28: Service decompositions with different cluster Factors (CF).

3. Map the NFs of the selected decomposition based on a backtracking mechanism

4.3.1 Implemented modules in ESCAPE

ESCAPE includes a **ros_mapping.py** module which contains classes to implement NFFG mapping functionality. In this section, we provide an overview of the existing classes in this module together with the additional classes implemented to integrate the explained embedding algorithm.

ros_mapping.py module. Contains classes which implement NFFG mapping functionality. These classes are illustrated in Fig. 29.

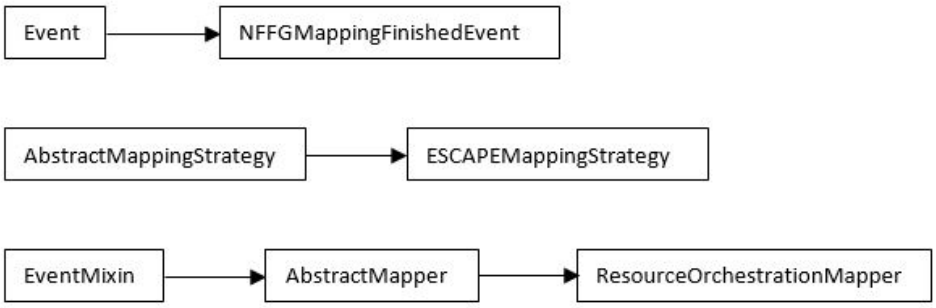


Figure 29: Existing classes in ros_mapping.py module.

ESCAPEMappingStrategy implements a default NFFG mapping algorithm of ESCAPE.

NFFGMappingFinishedEvent can signal the state of NFFG mapping.

ResourceOrchestrationMapper performs the supplementary tasks for NFFG mapping.

We have implemented a new mapping strategy class in the **ros_mapping** module which is shown in Fig. 30. The other 2 classes, i.e., NFFGMappingFinishedEvent and ResourceOrchestrationMapper, can be used without any change.

Figure 30: New class added to ros_mapping.py module.

iMindsMappingStrategy implements a NFFG mapping algorithm which jointly decomposes NFs and allocates resources to the NFs. This class is more detailed in Fig. 31.

class escape.orchest.ros_mapping.**iMindsMappingStrategy**

Bases: escape.util.mapping.AbstractMappingStrategy

Implements a strategy to map initial *NFFG* into extended *NFFG*. The mapping algorithm takes service decompositions into account.

__init__()

Init

classmethod map(*graph*, *resource*)

Mapping algorithm supporting NF decomposition for ESCAPEv2.

Parameters

- **graph** (*NFFG*) – Network Function Forwarding Graph
- **resource** (*NFFG*) – global virtual resource info

Returns mapped Network Function Forwarding Graph

Return type *NFFG*

Figure 31: Description of iMindsMappingStrategy class.

5 Conclusion

This document has delivered the final report on the UNIFY programmability framework. Besides the extensions and recently implemented components, we have focused on use cases which were highlighted first in [D3.3]. These examples illustrate several benefits and versatileness of our programmability framework and also the main advantages of the proposed UNIFY architecture.

More specifically, Section 2.1 has given a report on the latest updates and extensions on ESCAPE framework. The performance evaluation of the core embedding process of the orchestration framework has been provided in Section 2.2, while Section 2.3 has proposed an extension for VNF benchmarking. Several aspects of virtualization have been addressed in Section 3. In Section 3.1, the recent updates of our Virtualizer library has been summarized, while Section 3.2 has focused on the construction of the virtual resource views exposed towards upper level orchestrators in our recursive architecture.

Section 4 has been devoted to the use cases and related efforts towards the Integrated Prototype of UNIFY. In Section 4.1, the Elastic Router use case and related processes have been addressed. The integration of FlowNAC and the state migration process (FlowSNAC) has been presented in Section 4.2. And showing the benefits of our easily extensible, modular framework, the integration of a novel embedding algorithm taking decomposition choices into account has been discussed in Section 4.3.

The implemented software components of WP3 programmability framework have been released under open source licenses (mainly under Apache License Version 2) which enables future exploitation and further extensions in other EU projects or in future products.

5.1 Self-Assessment

Finally, we would like to provide a brief self-assessment of the research, design and proof of concept developments against the project's description of work:

Table 10: WP3 Status vs. DoW objectives

DoW Objective	Status
"Generic optimization framework which supports a variety of services and service chains, infrastructures, and objective functions"	Embedding algorithm
"We will develop tailored optimization techniques for the specific use cases."	Mapping algorithms takes parametric input on constraints and is generally applicable to different use-cases (see Elastic Router, FlowNAC)
"Our solution will jointly optimize the network and node resources, and across the network from the data centre over the access network to the network core. There are five main novelties: (1) joint optimization of network and nodes; (2) opportunity for in-network processing in "middle-boxes" and "nano data centres"; (3) time aspects and flexibility; (4) support for multi-stage mapping and chaining; (5) clustering of services and functions."	The joint domain abstraction and programmability for (1); Service Function Chaining programming for (2); seamless increment / decrement of SFCs and Cf-Or elasticity control interface for (3); recurring hierarchical domain abstraction and programming for (4) and automated decomposition at each layer for (5).
"A major objective of the optimization is the reduction of the complexity (e.g., need for configurations), rendering the management of the deployment less labour intensive. This reduces the OPEX/CAPEX costs."	Via full automation of service graph to service function chain deployment.
"We will implement the service orchestrator role of the overarching management and control: the orchestrator implements the optimization, based on the Network Information Base (NIB)."	The joint compute and network optimization is achieved at the UNIFY Resource Orchestrator; the Network Function Information Base at the Resoruce Orchestrator is a special NIB containing both VNF and networking resources, capabilities and VNF decomposition rules.
"Orchestrator is redundant and resilient, and also the location of the orchestrator itself: it may be realized close to the elements under its control to improve latency."	With the recurring orchestration layers we place orchestrators as close to the resources as possible (e.g., Universal Node). We have not addressed specific redundancy or resiliency questions of the orchestrators as independent orchestrator frameworks are specifically addressing those topics (e.g., ONOS Controller). We believe that our domain abstraction and programmability framework is applicable to such resilient architectures.
"To describe a programmability framework for the project scenario … define the necessary functionality at this framework and determine the primitives required."	We described extensively the generic programmability framework and its evolution throughout [D3.1][D3.2][D3.3][D3.4] and this document; discussing specific use-case like the Elastic Router or the FlowNAC; additionally, we showcased the UNIFY programmability framework at events like SIGCOMM 2014, 2015, Globecom 2015, ETSI From Research to Standardization event 2016 and the Bits-n-Bites of IETF96.
"This WP will also propose a subset of the framework and functionality that will be part of the prototype. The selected elements and building blocks will be implemented."	All the framework functions were implemented, tested and demonstrated.

A Demonstrating the multi-domain features and capabilities

In this section we describe a multi-domain demo setup as an illustration. The involved network domains are:

Mininet+Click domain with service access points (SAPs) running a terminal each. This domain runs ESCAPE as a local orchestrator.

SDN transport domain realized with hardware OpenFlow switches. The controller for these switches runs at the Global Orchestrator (POX).

Universal Node (UN) domain as a high performance telco datacenter with local orchestrator.

Docker domain as a lightweight execution environment.

OpenStack datacenter domain realized as a vanilla OS domain with a UNIFY virtualization adapter.

Important components of the demo are:

Global Orchestrator realized by ESCAPE, where our service request are sent to.

Local orchestrators of the domains.

Virtualizer library is used for the communication between Global Orchestrator and the domain local orchestrators. The information flow is bidirectional: the domains report their virtualized abstraction of the infrastructure, while the Global orchestrator deploys Network Functions and forwarding rules to realize the service.

The general structure of below service request sections (A.3 - A.7) will be:

- The service request is illustrated in a figure
- The service graph representation of the request is given in the ESCAPE JSON format, as well as in the virtualizer NFFG format, then NFFG is given in virtualizer format
- The result of Global Orchestrator's mapping of the request components to the infrastructure domains is illustrated in a figure
- Then the result of the mapping for each involved domain is given in the virtualizer NFFG format

A.1 Demo setup

The demo setup is depicted in Fig. 32. The abstract view of the same setup is shown in Fig. 33.

Figure 34 shows the communication diagram between the demo components. The solid arrows indicate communication during normal operation, while the dashed lines indicate extra communication added just for observation and documentation purposes. All the communication snapshots in this section can be mapped to this figure as:

step 8 : Listing 21 (single BiSBiS view) and Listing 22 (multi-BiSBiS view)

step 9 : Listings 26, 33, 40, 47, 54 in service graph format, and the corresponding Virtualizer xml format in Listings 27, 34, 41, 48, 55

step 11 : Listings 29, 36, 43, 50

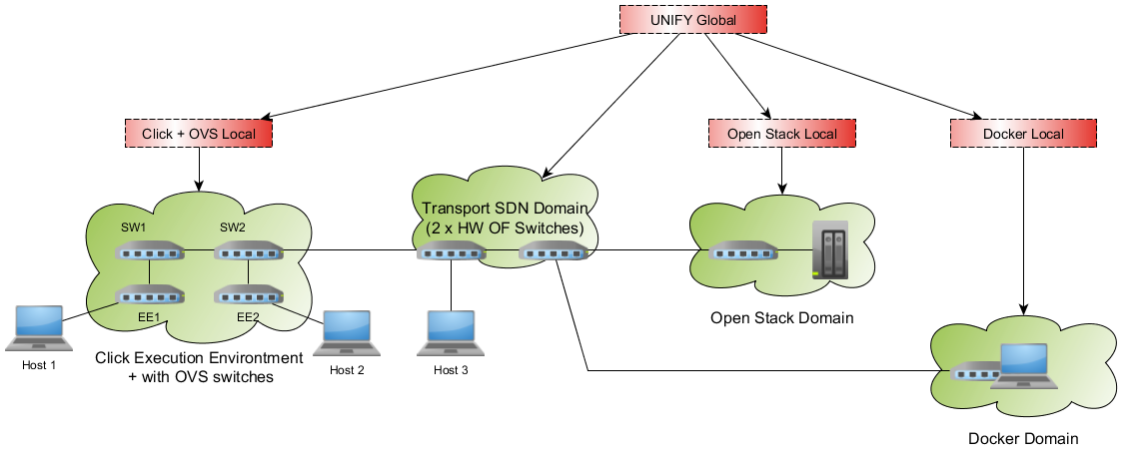


Figure 32: Infrastructure detailed view

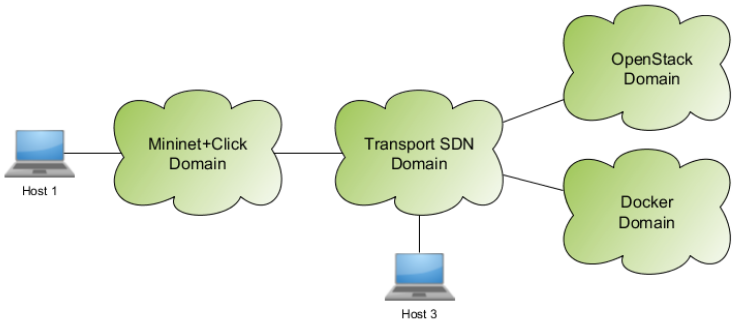


Figure 33: Infrastructure simple view

step 14 : Listings 28, 35

step 16 : Listings 49, 56

step 19 : Listings 30, 37, 44, 51, 57

step 21 : Listings 32, 39, 46, 53

step 23 : Listings 31, 38

step 25 : Listings 52, 58

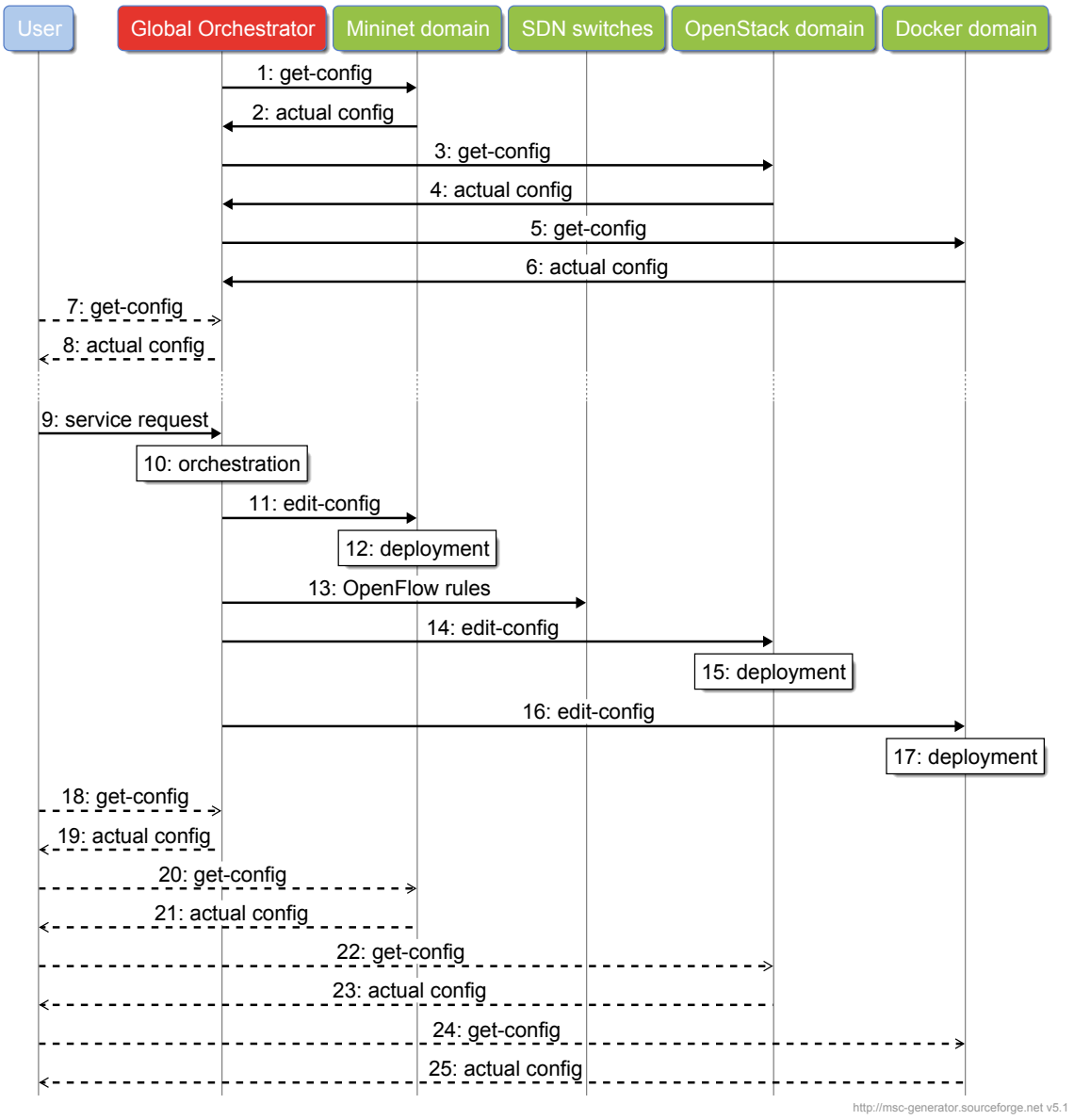


Figure 34: Message chart of the demo steps

A.2 Initial state of the domains

The initial ESCAPE (which is the global orchestrator) configuration according to the Virtualizer model is shown in Listing 21 where all underlying domains are merged (abstracted) to a single BiS-BiS, while in Listing 22 the separate domains are shown as separate nodes. Based on configuration, ESCAPE is capable to show both views. For simplicity reasons, we will present only the single BiS-BiS view in the rest of the document.

Listing 21: NFFG view of the initial global configuration - single node representation: xml view

```
<?xml version="1.0" ?>
<virtualizer>
  <id>SingleBiSBiS-NFFG</id>
  <name>Single-BiSBiS-View</name>
  <nodes>
    <node>
      <id>SingleBiSbiS</id>
      <name>SingleBiSBiS</name>
      <type>BiSBiS</type>
      <ports>
        <port>
          <id>port-SAP1</id>
          <name>SAP1</name>
          <port_type>port-sap</port_type>
        </port>
        <port>
          <id>port-SAP2</id>
          <name>SAP2</name>
          <port_type>port-sap</port_type>
        </port>
        <port>
          <id>port-SAP34</id>
          <name>SAP34</name>
          <port_type>port-sap</port_type>
        </port>
        <port>
          <id>port-SAP55</id>
          <name>SAP55</name>
          <port_type>port-sap</port_type>
        </port>
      </ports>
      <links>
        <link>
          <id>resource-link0</id>
          <src>../../../ports/port[id=port-SAP34]</src>
          <dst>../../../ports/port[id=port-SAP55]</dst>
          <resources>
            <delay>0.1</delay>
            <bandwidth>2880000.0</bandwidth>
          </resources>
        </link>
        <link>
          <id>resource-link1</id>
          <src>../../../ports/port[id=port-SAP34]</src>
          <dst>../../../ports/port[id=port-SAP1]</dst>
          <resources>
            <delay>0.1</delay>
            <bandwidth>2880000.0</bandwidth>
          </resources>
        </link>
        <link>
```

```
52            <id> resource —link2</id>
53            <src  >../../../  ports/port[id=port—SAP34]</src>
54            <dst  >../../../  ports/port[id=port—SAP2]</dst>
55            <resources>
56              <delay >0.1</ delay>
57              <bandwidth>2880000.0</bandwidth>
58            </resources>
59          </ link >
60          < link >
61            <id> resource —link3</id>
62            <src  >../../../  ports/port[id=port—SAP55]</src>
63            <dst  >../../../  ports/port[id=port—SAP1]</dst>
64            <resources>
65              <delay >0.1</ delay>
66              <bandwidth>2880000.0</bandwidth>
67            </resources>
68          </ link >
69          < link >
70            <id> resource —link4</id>
71            <src  >../../../  ports/port[id=port—SAP55]</src>
72            <dst  >../../../  ports/port[id=port—SAP2]</dst>
73            <resources>
74              <delay >0.1</ delay>
75              <bandwidth>2880000.0</bandwidth>
76            </resources>
77          </ link >
78          < link >
79            <id> resource —link5</id>
80            <src  >../../../  ports/port[id=port—SAP1]</src>
81            <dst  >../../../  ports/port[id=port—SAP2]</dst>
82            <resources>
83              <delay >0.1</ delay>
84              <bandwidth>2880000.0</bandwidth>
85            </resources>
86          </ link >
87        </ links >
88        <resources>
89          <cpu>29.0</cpu>
90          <mem>51082.0</mem>
91          <storage >1034.0</ storage>
92        </resources>
93        <metadata>
94          <key>bandwidth</key>
95          <value>2880000.0</value>
96        </metadata>
97        <metadata>
98          <key>delay</key>
99          <value >0.1</ value>
100       </metadata>
101       < capabilities >
102         <supported_NFs>
103           <node>
104             <id>dpi</id>
105             <type>dpi</type>
106           </node>
107           <node>
108             <id>headerCompressor</id>
109             <type>headerCompressor</type>
110           </node>
111           <node>
```

```
            <id>headerDecompressor</id>
            <type>headerDecompressor</type>
          </node>
          <node>
            <id> simpleForwarder </id>
            <type> simpleForwarder </type>
          </node>
          <node>
            <id>webserver</id>
            <type>webserver</type>
          </node>
        </supported_NFs>
      </ capabilities >
    </node>
  </nodes>
  < version >2016-02-24; compiled at  2016-03-18 19:56:13</ version >
</ virtualizer >
```

Listing 22: NFFG view of the initial global configuration - multiple nodes representation: xml view

```
<?xml  version = " 1.0 "  ?>
< virtualizer >
  <id>DoV</id>
  <name>DoV</name>
  <nodes>
    <node>
      <id>MT1</id>
      <name>MikroTik-SW-1</name>
      <type>SDN-SWITCH</type>
      <ports>
        <port>
          <id>1</id>
          <port_type>port-abstract</port_type>
        </port>
        <port>
          <id>2</id>
          <name>SAP14</name>
          <port_type>port-abstract</port_type>
          <sap>SAP14</sap>
        </port>
        <port>
          <id>3</id>
          <name>SAP55</name>
          <port_type>port-sap</port_type>
        </port>
        <port>
          <id>4</id>
          <port_type>port-abstract</port_type>
        </port>
      </ports>
      < links >
        < link >
          <id> resource -link0</id>
          <src  >../../..  ports/port[id =1]</ src>
          <dst  >../../..  ports/port[id =2]</ dst>
          <resources>
            <delay >0.2</ delay>
            <bandwidth>4000.0</bandwidth>
          </resources>
        </ link >
```

```
        <link>
          <id>resource—link1</id>
          <src>../../../ports/port[id=1]</src>
          <dst>../../../ports/port[id=3]</dst>
          <resources>
            <delay>0.2</delay>
            <bandwidth>4000.0</bandwidth>
          </resources>
        </link>
        <link>
          <id>resource—link2</id>
          <src>../../../ports/port[id=1]</src>
          <dst>../../../ports/port[id=4]</dst>
          <resources>
            <delay>0.2</delay>
            <bandwidth>4000.0</bandwidth>
          </resources>
        </link>
        <link>
          <id>resource—link3</id>
          <src>../../../ports/port[id=2]</src>
          <dst>../../../ports/port[id=3]</dst>
          <resources>
            <delay>0.2</delay>
            <bandwidth>4000.0</bandwidth>
          </resources>
        </link>
        <link>
          <id>resource—link4</id>
          <src>../../../ports/port[id=2]</src>
          <dst>../../../ports/port[id=4]</dst>
          <resources>
            <delay>0.2</delay>
            <bandwidth>4000.0</bandwidth>
          </resources>
        </link>
        <link>
          <id>resource—link5</id>
          <src>../../../ports/port[id=3]</src>
          <dst>../../../ports/port[id=4]</dst>
          <resources>
            <delay>0.2</delay>
            <bandwidth>4000.0</bandwidth>
          </resources>
        </link>
      </links>
      <metadata>
        <key>bandwidth</key>
        <value>4000.0</value>
      </metadata>
      <metadata>
        <key>delay</key>
        <value>0.2</value>
      </metadata>
    </node>
    <node>
      <id>MT2</id>
      <name>MikroTik—SW—2</name>
      <type>SDN—SWITCH</type>
      <ports>
```

```
        <port>
          <id>1</id>
          <port_type>port-abstract</port_type>
        </port>
        <port>
          <id>2</id>
          <name>SAP24</name>
          <port_type>port-abstract</port_type>
          <sap>SAP24</sap>
        </port>
        <port>
          <id>3</id>
          <name>SAP34</name>
          <port_type>port-sap</port_type>
        </port>
        <port>
          <id>4</id>
          <port_type>port-abstract</port_type>
        </port>
      </ports>
      <links>
        <link>
          <id>resource-link0</id>
          <src>../../../ports/port[id=1]</src>
          <dst>../../../ports/port[id=2]</dst>
          <resources>
            <delay>0.2</delay>
            <bandwidth>4000.0</bandwidth>
          </resources>
        </link>
        <link>
          <id>resource-link1</id>
          <src>../../../ports/port[id=1]</src>
          <dst>../../../ports/port[id=3]</dst>
          <resources>
            <delay>0.2</delay>
            <bandwidth>4000.0</bandwidth>
          </resources>
        </link>
        <link>
          <id>resource-link2</id>
          <src>../../../ports/port[id=1]</src>
          <dst>../../../ports/port[id=4]</dst>
          <resources>
            <delay>0.2</delay>
            <bandwidth>4000.0</bandwidth>
          </resources>
        </link>
        <link>
          <id>resource-link3</id>
          <src>../../../ports/port[id=2]</src>
          <dst>../../../ports/port[id=3]</dst>
          <resources>
            <delay>0.2</delay>
            <bandwidth>4000.0</bandwidth>
          </resources>
        </link>
        <link>
          <id>resource-link4</id>
          <src>../../../ports/port[id=2]</src>
```

```
            <dst  >../../..  ports/port[id=4]</dst>
            <resources>
              <delay>0.2</delay>
              <bandwidth>4000.0</bandwidth>
            </resources>
          </link>
          <link>
            <id>resource-link5</id>
            <src  >../../..  ports/port[id=3]</src>
            <dst  >../../..  ports/port[id=4]</dst>
            <resources>
              <delay>0.2</delay>
              <bandwidth>4000.0</bandwidth>
            </resources>
          </link>
        </links>
        <metadata>
          <key>bandwidth</key>
          <value>4000.0</value>
        </metadata>
        <metadata>
          <key>delay</key>
          <value>0.2</value>
        </metadata>
      </node>
      <node>
        <id>SingleBiSbiS@REMOTE</id>
        <name>SingleBiSBiS</name>
        <type>BiSBiS</type>
        <ports>
          <port>
            <id>port-SAP1</id>
            <name>SAP1</name>
            <port_type>port-sap</port_type>
          </port>
          <port>
            <id>port-SAP14</id>
            <name>SAP14</name>
            <port_type>port-abstract</port_type>
            <sap>SAP14</sap>
          </port>
          <port>
            <id>port-SAP2</id>
            <name>SAP2</name>
            <port_type>port-sap</port_type>
          </port>
        </ports>
        <links>
          <link>
            <id>resource-link0</id>
            <src  >../../..  ports/port[id=port-SAP1]</src>
            <dst  >../../..  ports/port[id=port-SAP14]</dst>
            <resources>
              <delay>0.2</delay>
              <bandwidth>160000.0</bandwidth>
            </resources>
          </link>
          <link>
            <id>resource-link1</id>
            <src  >../../..  ports/port[id=port-SAP1]</src>
```

```
            <dst >../../../ ports/port[id=port-SAP2]</dst>
            <resources>
              <delay>0.2</delay>
              <bandwidth>160000.0</bandwidth>
            </resources>
          </link>
          <link>
            <id>resource-link2</id>
            <src >../../../ ports/port[id=port-SAP14]</src>
            <dst >../../../ ports/port[id=port-SAP2]</dst>
            <resources>
              <delay>0.2</delay>
              <bandwidth>160000.0</bandwidth>
            </resources>
          </link>
        </links>
        <resources>
          <cpu>10.0</cpu>
          <mem>10.0</mem>
          <storage>10.0</storage>
        </resources>
        <metadata>
          <key>bandwidth</key>
          <value>160000.0</value>
        </metadata>
        <metadata>
          <key>delay</key>
          <value>0.2</value>
        </metadata>
        <capabilities>
          <supported_NFs>
            <node>
              <id>headerCompressor</id>
              <type>headerCompressor</type>
            </node>
            <node>
              <id>headerDecompressor</id>
              <type>headerDecompressor</type>
            </node>
            <node>
              <id>simpleForwarder</id>
              <type>simpleForwarder</type>
            </node>
          </supported_NFs>
        </capabilities>
      </node>
      <node>
        <id>UUID-OpenStack-01-pack@OPENSTACK</id>
        <name>single Bis-Bis node representing the whole OpenStack domain-packaged</name>
        <type>BisBis</type>
        <ports>
          <port>
            <id>0</id>
            <name>SAP24</name>
            <port_type>port-abstract</port_type>
            <sap>SAP24</sap>
          </port>
        </ports>
        <links>
          <link>
```

```
            <id> resource —link</id>
            <src  >../../../  ports/port[id=0]</src>
            <dst  >../../../  ports/port[id=0]</dst>
            <resources>
              <delay>None</delay>
              <bandwidth>None</bandwidth>
            </resources>
          </link>
        </links>
        <resources>
          <cpu>19.0</cpu>
          <mem>51072.0</mem>
          <storage>1024.0</storage>
        </resources>
        <capabilities>
          <supported_NFs>
            <node>
              <id>dpi</id>
              <type>dpi</type>
            </node>
            <node>
              <id>webserver</id>
              <type>webserver</type>
            </node>
          </supported_NFs>
        </capabilities>
      </node>
    </nodes>
    <links>
      <link>
        <id> inter —domain—link—SAP14</id>
        <src  >../../../  nodes/node[id=SingleBiSbiS@REMOTE]/ports/port[id=port—SAP14]</src>
        <dst  >../../../  nodes/node[id=MT1]/ports/port[id=2]</dst>
      </link>
      <link>
        <id> inter —domain—link—SAP14—back</id>
        <src  >../../../  nodes/node[id=MT1]/ports/port[id=2]</src>
        <dst  >../../../  nodes/node[id=SingleBiSbiS@REMOTE]/ports/port[id=port—SAP14]</dst>
      </link>
      <link>
        <id> inter —domain—link—SAP24</id>
        <src  >../../../  nodes/node[id=UUID—OpenStack—01—pack@OPENSTACK]/ports/port[id=0]</src>
        <dst  >../../../  nodes/node[id=MT2]/ports/port[id=2]</dst>
      </link>
      <link>
        <id> inter —domain—link—SAP24—back</id>
        <src  >../../../  nodes/node[id=MT2]/ports/port[id=2]</src>
        <dst  >../../../  nodes/node[id=UUID—OpenStack—01—pack@OPENSTACK]/ports/port[id=0]</dst>
      </link>
      <link>
        <id>sdn—link1</id>
        <src  >../../../  nodes/node[id=MT1]/ports/port[id=1]</src>
        <dst  >../../../  nodes/node[id=MT2]/ports/port[id=1]</dst>
        <resources>
          <delay>0.1</delay>
          <bandwidth>1000.0</bandwidth>
        </resources>
      </link>
      <link>
        <id>sdn—link1—back</id>
```

```
        <src  >../../../  nodes/node[id=MT2]/ports/port[id =1]</ src>
        <dst  >../../../  nodes/node[id=MT1]/ports/port[ id =1]</ dst>
        <resources>
          <delay >0.1</ delay>
          <bandwidth>1000.0</bandwidth>
        </resources>
      </ link >
    </ links >
    < version >2016—02—24; compiled at  2016—03—18  19:56:13</ version >
</  virtualizer >
```

The initial configuration of the OpenStack configuration is shown Listing 23, the Mininet configuration in Listing 24, and the Docker domain configuration in Listing 25.

Listing 23: NFFG of Initial OpenStack configuration: xml view

```
<?xml  version =" 1.0 "  ?>
< virtualizer >
  <id>UUID—ETH—001—pack</id>
  <name>ETH OpenStack domain—packaged</name>
  <nodes>
    <node>
      <id>UUID—OpenStack—01—pack</id>
      <name>single  Bis—Bis node  representing  the  whole  OpenStack  domain—packaged</name>
      <type> BisBis </type>
      <ports>
        <port>
          <id>0</id>
          <name>SAP24</name>
          <port_type>port—sap</port_type>
          <sap>SAP24</sap>
        </port>
      </ports>
      <resources>
        <cpu>20</cpu>
        <mem>51200</mem>
        <storage>1024</ storage>
      </resources>
      < capabilities >
        <supported_NFs>
          <node>
            <id>dpi</id>
            <name>dpi4</name>
            <type>dpi</type>
            <ports>
              <port>
                <id>1</ id>
                <name>input</name>
                <port_type>port—abstract</port_type>
              </port>
              <port>
                <id>2</ id>
                <name>output</name>
                <port_type>port—abstract</port_type>
              </port>
            </ports>
            < links >
              < link >
                <id>l1</id>
                <name> internal1 </name>
                <src  >../../../  ports/port[ id =1]</ src>
```

```
                <dst>../../../ports/port[id=2]</dst>
              </link>
            </links>
            <resources>
              <cpu>1</cpu>
              <mem>128</mem>
              <storage>1</storage>
            </resources>
          </node>
          <node>
            <id>webserver</id>
            <name>webserver4</name>
            <type>webserver</type>
            <ports>
              <port>
                <id>0</id>
                <name>in-out</name>
                <port_type>port-abstract</port_type>
              </port>
            </ports>
            <resources>
              <cpu>1</cpu>
              <mem>128</mem>
              <storage>1</storage>
            </resources>
          </node>
        </supported_NFs>
      </capabilities>
    </node>
  </nodes>
  <version>2016-02-24; compiled at 2016-03-18 19:56:13</version>
</virtualizer>
```

Listing 24: NFFG of Initial Mininet configuration: xml view

```
<?xml version="1.0"?>
<virtualizer>
  <id>SingleBiSBiS-NFFG</id>
  <name>Single-BiSBiS-View</name>
  <nodes>
    <node>
      <id>SingleBiSBiS</id>
      <name>SingleBiSBiS</name>
      <type>BiSBiS</type>
      <ports>
        <port>
          <id>port-SAP1</id>
          <name>SAP1</name>
          <port_type>port-sap</port_type>
        </port>
        <port>
          <id>port-SAP14</id>
          <name>SAP14</name>
          <port_type>port-sap</port_type>
        </port>
        <port>
          <id>port-SAP2</id>
          <name>SAP2</name>
          <port_type>port-sap</port_type>
        </port>
```

```
26        </ports>
27        < links >
28          < link >
29            <id> resource —link0</ id >
30            <src  >../../../  ports/port[id=port—SAP14]</src>
31            <dst  >../../../  ports/port[id=port—SAP1]</dst>
32            <resources>
33              <delay >0.2</ delay>
34              <bandwidth>1120000.0</bandwidth>
35            </resources>
36          </ link >
37          < link >
38            <id> resource —link1</ id >
39            <src  >../../../  ports/port[id=port—SAP14]</src>
40            <dst  >../../../  ports/port[id=port—SAP2]</dst>
41            <resources>
42              <delay >0.2</ delay>
43              <bandwidth>1120000.0</bandwidth>
44            </resources>
45          </ link >
46          < link >
47            <id> resource —link2</ id >
48            <src  >../../../  ports/port[id=port—SAP1]</src>
49            <dst  >../../../  ports/port[id=port—SAP2]</dst>
50            <resources>
51              <delay >0.2</ delay>
52              <bandwidth>1120000.0</bandwidth>
53            </resources>
54          </ link >
55        </ links >
56        <resources>
57          <cpu>10.0</cpu>
58          <mem>10.0</mem>
59          <storage >10.0</ storage>
60        </resources>
61        <metadata>
62          <key>bandwidth</key>
63          <value>1120000.0</value>
64        </metadata>
65        <metadata>
66          <key>delay</key>
67          <value >0.2</ value>
68        </metadata>
69        < capabilities >
70          <supported_NFs>
71            <node>
72              <id>headerCompressor</id>
73              <type>headerCompressor</type>
74            </node>
75            <node>
76              <id>headerDecompressor</id>
77              <type>headerDecompressor</type>
78            </node>
79            <node>
80              <id> simpleForwarder </id>
81              <type> simpleForwarder </type>
82            </node>
83          </supported_NFs>
84        </ capabilities >
85      </node>
```

```
86    </nodes>
87    < version >2016-02-24; compiled at  2016-03-18 19:56:13</ version >
88  </ virtualizer >
```

Listing 25: NFFG of Initial Docker configuration: xml view

```
1  <?xml  version =" 1.0 "  ?>
2  < virtualizer >
3    <id>DOCKER</id>
4    <name>Docker domain</name>
5    <nodes>
6      <node>
7        <id>DOCKER_HOST</id>
8        <name>atitude-Linux</name>
9        <type> BiSBiS </type>
10       <ports>
11         <port>
12           <id>eth0</id>
13           <name>SAP34</name>
14           <port_type>port-sap</port_type>
15           <sap>SAP34</sap>
16         </port>
17       </ports>
18       <resources>
19         <cpu>4</cpu>
20         <mem>50721017856</mem>
21         <storage>25010147328</storage>
22       </resources>
23       < capabilities >
24         <supported_NFs>
25           <node>
26             <id> dockernf </id>
27             <name>dockernf</name>
28             <type> bridge </type>
29             <ports>
30               <port>
31                 <id>eth0</id>
32                 <name>mgmt</name>
33                 <port_type> public -ports</port_type>
34                 <metadata>
35                   <key>  localports  </key>
36                   <value>22</value>
37                 </metadata>
38               </port>
39               <port>
40                 <id>eth1</id>
41                 <name>input</name>
42                 <port_type>port- abstract </port_type>
43               </port>
44               <port>
45                 <id>eth2</id>
46                 <name>output</name>
47                 <port_type>port- abstract </port_type>
48               </port>
49             </ports>
50             <resources>
51               <cpu>1</cpu>
52               <mem>128</mem>
53               <storage>1</ storage>
54             </resources>
```

```
55              </node>
56            </supported_NFs>
57          </ capabilities >
58        </node>
59      </nodes>
60      < version >2016-02-24; compiled at  2016-03-18  19:56:13</ version >
61  </ virtualizer >
```

A.3 Service Request 1: Web server

In the first example we request a web server to be connected to SAP1 (Host 1, user 1, etc.). The first service request is depicted in Fig. 35, and formalized in Listing 26 to be sent to ESCAPE.

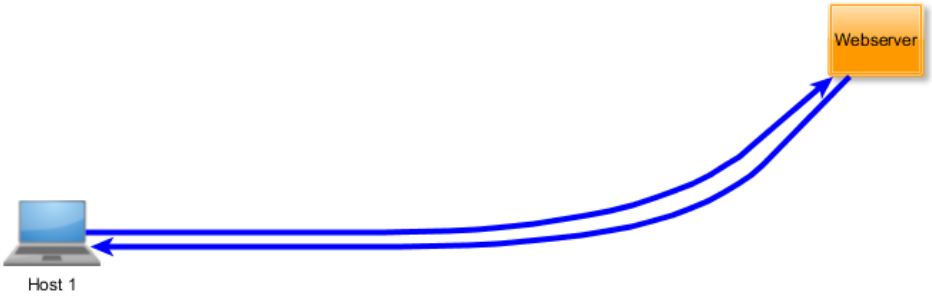


Figure 35: request1

Listing 26: First service request (service graph) to ESCAPE

```
{
  "parameters": {
    "id": "EWSDN-demo-req1",
    "name": "EWSDN-2web-2SAP-req",
    "version": "1.0"
  },
  "node_nfs": [
    {
      "id": "webserver1",
      "name": "webserver1",
      "ports": [
        {
          "id": 0
        }
      ],
      "functional_type": "webserver",
      "specification": {
        "resources": {
          "cpu": 1,
          "mem": 1,
          "storage": 0
        }
      }
    }
  ],
  "node_saps": [
    {
      "id": "sap1",
      "name": "SAP1",
      "ports": [
        {
          "id": 1
        }
      ]
    },
    {
      "id": "sap2",
      "name": "SAP2",
      "ports": [
```

```
        {
          "id": 1
        }
      ]
    }
  ],
  "edge_sg_nexthops": [
    {
      "id": 12,
      "src_node": "webserver1",
      "src_port": 0,
      "dst_node": "sap1",
      "dst_port": 1
    },
    {
      "id": 11,
      "src_node": "sap1",
      "src_port": 1,
      "dst_node": "webserver1",
      "dst_port": 0
    }
  ],
  "edge_reqs": [
    {
      "id": 51282000,
      "src_node": "sap1",
      "src_port": 1,
      "dst_node": "sap1",
      "dst_port": 1,
      "delay": 100,
      "bandwidth": 1,
      "sg_path": [
        11,
        12
      ]
    }
  ]
}
```

The above service request can be formalized with the Virtualizer model as described in Listing 27.

Listing 27: NFFG of First request (Virtualizer) to ESCAPE: xml view

```
<?xml version="1.0" ?>
<virtualizer>
  <nodes>
    <node>
      <id>SingleBiSbiS</id>
      <NF_instances>
        <node operation="create">
          <id>webserver1</id>
          <name>webserver1</name>
          <type>webserver</type>
          <ports>
            <port>
              <id>0</id>
              <port_type>port-abstract</port_type>
            </port>
          </ports>
          <resources>
            <cpu>1.0</cpu>
```

```
            <mem>1.0</mem>
            <storage>0.0</storage>
          </resources>
        </node>
      </NF_instances>
      <flowtable>
        <flowentry operation="create">
          <id>ESCAPE-flowentry11</id>
          <name>sg_hop:11</name>
          <priority>100</priority>
          <port>../../../ports/port[id=port-SAP1]</port>
          <out>../../../NF_instances/node[id=webserver1]/ports/port[id=0]</out>
          <resources>
            <bandwidth>1.0</bandwidth>
          </resources>
        </flowentry>
        <flowentry operation="create">
          <id>ESCAPE-flowentry12</id>
          <name>sg_hop:12</name>
          <priority>100</priority>
          <port>../../../NF_instances/node[id=webserver1]/ports/port[id=0]</port>
          <out>../../../ports/port[id=port-SAP1]</out>
          <resources>
            <bandwidth>1.0</bandwidth>
          </resources>
        </flowentry>
      </flowtable>
    </node>
  </nodes>
  <version>2016-02-24; compiled at 2016-03-18 19:56:13</version>
</virtualizer>
```

The outcome of the global orchestration is shown in Fig. 36.

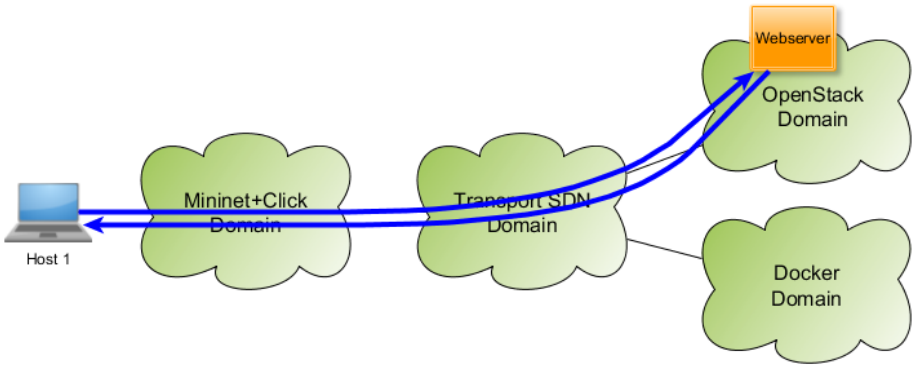


Figure 36: deployed1

The configuration messages sent by the global orchestrator to the OpenStack domain is visible in Listing 28, and to the Mininet domain in Listing 29.

Listing 28: NFFG of First step configuration message to OpenStack domain: xml view

```
<?xml version="1.0" ?>
<virtualizer>
  <nodes>
    <node>
      <id>UUID-OpenStack-01-pack</id>
```

```
      <NF_instances>
        <node operation="create">
          <id>webserver1</id>
          <name>webserver1</name>
          <type>webserver</type>
          <ports>
            <port>
              <id>0</id>
              <port_type>port-abstract</port_type>
            </port>
          </ports>
          <resources>
            <cpu>1.0</cpu>
            <mem>1.0</mem>
            <storage>0.0</storage>
          </resources>
        </node>
      </NF_instances>
      <flowtable>
        <flowentry operation="create">
          <id>ESCAPE-flowentry11</id>
          <name>sg_hop:11</name>
          <priority>100</priority>
          <port>../../../ports/port[id=0]</port>
          <match>dl_tag=0x000b</match>
          <action>pop_tag</action>
          <out>../../../NF_instances/node[id=webserver1]/ports/port[id=0]</out>
        </flowentry>
        <flowentry operation="create">
          <id>ESCAPE-flowentry12</id>
          <name>sg_hop:12</name>
          <priority>100</priority>
          <port>../../../NF_instances/node[id=webserver1]/ports/port[id=0]</port>
          <action>push_tag:0x000c</action>
          <out>../../../ports/port[id=0]</out>
        </flowentry>
      </flowtable>
    </node>
  </nodes>
  <version>2016-02-24; compiled at 2016-03-18 19:56:13</version>
</virtualizer>
```

Listing 29: NFFG of First step configuration message to Mininet domain: xml view

```
<?xml version="1.0"?>
<virtualizer>
  <nodes>
    <node>
      <id>SingleBiSBiS</id>
      <flowtable>
        <flowentry operation="create">
          <id>ESCAPE-flowentry11</id>
          <name>sg_hop:11</name>
          <priority>100</priority>
          <port>../../../ports/port[id=port-SAP1]</port>
          <out>../../../ports/port[id=port-SAP14]</out>
        </flowentry>
        <flowentry operation="create">
          <id>ESCAPE-flowentry12</id>
          <name>sg_hop:12</name>
```

```
            < priority >100</ priority >
            <port >../../../ ports/port[id=port-SAP14]</port>
            <out >../../../ ports/port[id=port-SAP1]</out>
          </flowentry>
        </flowtable>
      </node>
    </nodes>
    < version >2016-02-24; compiled at 2016-03-18 19:56:13</ version >
</ virtualizer >
```

The result global configurations configuration is showed in Listing 30, the OpenStack configuration in Listing 31, and the Mininet configuration in Listing 32.

Listing 30: NFFG of First step result: global configuration: xml view

```
<?xml version =" 1.0 " ?>
< virtualizer >
  <id> SingleBiSBiS -NFFG</id>
  <name>Single-BiSBiS-View</name>
  <nodes>
    <node>
      <id> SingleBiSbiS </id>
      <name> SingleBiSBiS </name>
      <type> BiSBiS </type>
      <ports>
        <port>
          <id>port-SAP1</id>
          <name>SAP1</name>
          <port_type>port-sap</port_type>
        </port>
        <port>
          <id>port-SAP2</id>
          <name>SAP2</name>
          <port_type>port-sap</port_type>
        </port>
        <port>
          <id>port-SAP34</id>
          <name>SAP34</name>
          <port_type>port-sap</port_type>
        </port>
        <port>
          <id>port-SAP55</id>
          <name>SAP55</name>
          <port_type>port-sap</port_type>
        </port>
      </ports>
      < links >
        < link >
          <id> resource -link0</id>
          <src >../../../ ports/port[id=port-SAP34]</src>
          <dst >../../../ ports/port[id=port-SAP55]</dst>
          <resources>
            <delay>0</delay>
            <bandwidth>3200000.0</bandwidth>
          </resources>
        </ link >
        < link >
          <id> resource -link1</id>
          <src >../../../ ports/port[id=port-SAP34]</src>
          <dst >../../../ ports/port[id=port-SAP1]</dst>
          <resources>
```

```
            <delay>0</delay>
            <bandwidth>3200000.0</bandwidth>
          </resources>
        </link>
        <link>
          <id>resource—link2</id>
          <src>../../../ports/port[id=port—SAP34]</src>
          <dst>../../../ports/port[id=port—SAP2]</dst>
          <resources>
            <delay>0</delay>
            <bandwidth>3200000.0</bandwidth>
          </resources>
        </link>
        <link>
          <id>resource—link3</id>
          <src>../../../ports/port[id=port—SAP55]</src>
          <dst>../../../ports/port[id=port—SAP1]</dst>
          <resources>
            <delay>0</delay>
            <bandwidth>3200000.0</bandwidth>
          </resources>
        </link>
        <link>
          <id>resource—link4</id>
          <src>../../../ports/port[id=port—SAP55]</src>
          <dst>../../../ports/port[id=port—SAP2]</dst>
          <resources>
            <delay>0</delay>
            <bandwidth>3200000.0</bandwidth>
          </resources>
        </link>
        <link>
          <id>resource—link5</id>
          <src>../../../ports/port[id=port—SAP1]</src>
          <dst>../../../ports/port[id=port—SAP2]</dst>
          <resources>
            <delay>0</delay>
            <bandwidth>3200000.0</bandwidth>
          </resources>
        </link>
      </links>
      <resources>
        <cpu>29.0</cpu>
        <mem>51082.0</mem>
        <storage>1034.0</storage>
      </resources>
      <metadata>
        <key>bandwidth</key>
        <value>3200000.0</value>
      </metadata>
      <metadata>
        <key>delay</key>
        <value>0</value>
      </metadata>
      <NF_instances>
        <node>
          <id>webserver1</id>
          <name>webserver1</name>
          <type>webserver</type>
          <ports>
```

```
            <port>
              <id>0</id>
              <port_type>port-abstract</port_type>
            </port>
          </ports>
          <resources>
            <cpu>1.0</cpu>
            <mem>1.0</mem>
            <storage>0.0</storage>
          </resources>
        </node>
      </NF_instances>
      < capabilities >
        <supported_NFs>
          <node>
            <id>dpi</id>
            <type>dpi</type>
          </node>
          <node>
            <id>headerCompressor</id>
            <type>headerCompressor</type>
          </node>
          <node>
            <id>headerDecompressor</id>
            <type>headerDecompressor</type>
          </node>
          <node>
            <id> simpleForwarder </id>
            <type> simpleForwarder </type>
          </node>
          <node>
            <id>webserver</id>
            <type>webserver</type>
          </node>
        </supported_NFs>
      </ capabilities >
      <flowtable>
        <flowentry>
          <id>ESCAPE-flowentry11</id>
          <name>sg_hop:11</name>
          < priority >100</ priority >
          <port >../../../ ports/port[id=port-SAP1]</port>
          <out >../../../ NF_instances/node[id=webserver1]/ports/port[id=0]</out>
          <resources>
            <bandwidth>1.0</bandwidth>
          </resources>
        </flowentry>
        <flowentry>
          <id>ESCAPE-flowentry12</id>
          <name>sg_hop:12</name>
          < priority >100</ priority >
          <port >../../../ NF_instances/node[id=webserver1]/ports/port[id=0]</port>
          <out >../../../ ports/port[id=port-SAP1]</out>
          <resources>
            <bandwidth>1.0</bandwidth>
          </resources>
        </flowentry>
      </flowtable>
    </node>
  </nodes>
```

```
    < version >2016-02-24; compiled at  2016-03-18  19:56:13</ version >
</ virtualizer >
```

Listing 31: NFFG of First step result: OpenStack configuration: xml view

```
<?xml  version =" 1.0 "  ?>
< virtualizer >
  <id>UUID-ETH-001-pack</id>
  <name>ETH OpenStack domain-packaged</name>
  <nodes>
    <node>
      <id>UUID-OpenStack-01-pack</id>
      <name>single  Bis-Bis node  representing  the  whole  OpenStack  domain-packaged</name>
      <type> BisBis </type>
      <ports>
        <port>
          <id>0</id>
          <name>SAP24</name>
          <port_type>port-sap</port_type>
          <sap>SAP24</sap>
        </port>
      </ports>
      <resources>
        <cpu>19</cpu>
        <mem>51072</mem>
        <storage>1024</storage>
      </resources>
      <NF_instances>
        <node>
          <id>webserver1</id>
          <name>webserver1</name>
          <type>webserver</type>
          <ports>
            <port>
              <id>0</id>
              <port_type>port-abstract</port_type>
            </port>
          </ports>
          <resources>
            <cpu>1.0</cpu>
            <mem>1.0</mem>
            <storage>0.0</storage>
          </resources>
        </node>
      </NF_instances>
      < capabilities >
        <supported_NFs>
          <node>
            <id>dpi</id>
            <name>dpi4</name>
            <type>dpi</type>
            <ports>
              <port>
                <id>1</id>
                <name>input</name>
                <port_type>port-abstract</port_type>
              </port>
              <port>
                <id>2</id>
                <name>output</name>
```

```
                <port_type>port-abstract</port_type>
              </port>
            </ports>
            <links>
              <link>
                <id>l1</id>
                <name>internal1</name>
                <src>../../../ports/port[id=1]</src>
                <dst>../../../ports/port[id=2]</dst>
              </link>
            </links>
            <resources>
              <cpu>1</cpu>
              <mem>128</mem>
              <storage>1</storage>
            </resources>
          </node>
          <node>
            <id>webserver</id>
            <name>webserver4</name>
            <type>webserver</type>
            <ports>
              <port>
                <id>0</id>
                <name>in-out</name>
                <port_type>port-abstract</port_type>
              </port>
            </ports>
            <resources>
              <cpu>1</cpu>
              <mem>128</mem>
              <storage>1</storage>
            </resources>
          </node>
        </supported_NFs>
      </capabilities>
      <flowtable>
        <flowentry>
          <id>ESCAPE-flowentry11</id>
          <name>sg_hop:11</name>
          <priority>100</priority>
          <port>../../../ports/port[id=0]</port>
          <match>dl_tag=0x000b</match>
          <action>pop_tag</action>
          <out>../../../NF_instances/node[id=webserver1]/ports/port[id=0]</out>
          <resources>
            <bandwidth>1.0</bandwidth>
          </resources>
        </flowentry>
        <flowentry>
          <id>ESCAPE-flowentry12</id>
          <name>sg_hop:12</name>
          <priority>100</priority>
          <port>../../../NF_instances/node[id=webserver1]/ports/port[id=0]</port>
          <action>push_tag:0x000c</action>
          <out>../../../ports/port[id=0]</out>
          <resources>
            <bandwidth>1.0</bandwidth>
          </resources>
        </flowentry>
```

```
        </flowtable>
      </node>
    </nodes>
    < version >2016-02-24; compiled at  2016-03-18 19:56:13</ version >
</ virtualizer >
```

Listing 32: NFFG of First step result: Mininet configuration: xml view

```
<?xml  version =" 1.0 "  ?>
< virtualizer >
  <id> SingleBiSBiS -NFFG</id>
  <name>Single-BiSBiS-View</name>
  <nodes>
    <node>
      <id> SingleBiSBiS </id>
      <name>SingleBiSBiS</name>
      <type>BiSBiS</type>
      <ports>
        <port>
          <id>port-SAP1</id>
          <name>SAP1</name>
          <port_type>port-sap</port_type>
        </port>
        <port>
          <id>port-SAP14</id>
          <name>SAP14</name>
          <port_type>port-sap</port_type>
          <sap>SAP14</sap>
        </port>
        <port>
          <id>port-SAP2</id>
          <name>SAP2</name>
          <port_type>port-sap</port_type>
        </port>
      </ports>
      < links >
        < link >
          <id> resource -link0</id>
          <src  >../../../  ports/port[id=port-SAP14]</src>
          <dst  >../../../  ports/port[id=port-SAP1]</dst>
          <resources>
            <delay >0.2</ delay>
            <bandwidth>160000.0</bandwidth>
          </resources>
        </ link >
        < link >
          <id> resource -link1</id>
          <src  >../../../  ports/port[id=port-SAP14]</src>
          <dst  >../../../  ports/port[id=port-SAP2]</dst>
          <resources>
            <delay >0.2</ delay>
            <bandwidth>160000.0</bandwidth>
          </resources>
        </ link >
        < link >
          <id> resource -link2</id>
          <src  >../../../  ports/port[id=port-SAP1]</src>
          <dst  >../../../  ports/port[id=port-SAP2]</dst>
          <resources>
            <delay >0.2</ delay>
```

```
            <bandwidth>160000.0</bandwidth>
          </resources>
        </link>
      </links>
      <resources>
        <cpu>10.0</cpu>
        <mem>10.0</mem>
        <storage>10.0</storage>
      </resources>
      <metadata>
        <key>bandwidth</key>
        <value>160000.0</value>
      </metadata>
      <metadata>
        <key>delay</key>
        <value>0.2</value>
      </metadata>
      <capabilities>
        <supported_NFs>
          <node>
            <id>headerCompressor</id>
            <type>headerCompressor</type>
          </node>
          <node>
            <id>headerDecompressor</id>
            <type>headerDecompressor</type>
          </node>
          <node>
            <id>simpleForwarder</id>
            <type>simpleForwarder</type>
          </node>
        </supported_NFs>
      </capabilities>
      <flowtable>
        <flowentry>
          <id>ESCAPE-flowentry11</id>
          <name>sg_hop:11</name>
          <priority>100</priority>
          <port>../../../ports/port[id=port-SAP1]</port>
          <out>../../../ports/port[id=port-SAP14]</out>
          <resources>
            <bandwidth>0.0</bandwidth>
          </resources>
        </flowentry>
        <flowentry>
          <id>ESCAPE-flowentry12</id>
          <name>sg_hop:12</name>
          <priority>100</priority>
          <port>../../../ports/port[id=port-SAP14]</port>
          <out>../../../ports/port[id=port-SAP1]</out>
          <resources>
            <bandwidth>1.0</bandwidth>
          </resources>
        </flowentry>
      </flowtable>
    </node>
  </nodes>
  <version>2016-02-24; compiled at 2016-03-18 19:56:13</version>
</virtualizer>
```

A.4 Service Request 2: DPI added

In the second example we extend the existing setup with requesting a DPI to the backward path from the web server. The second service request is depicted in Fig. 37, and formalized in Listing 33 to be sent to ESCAPE.

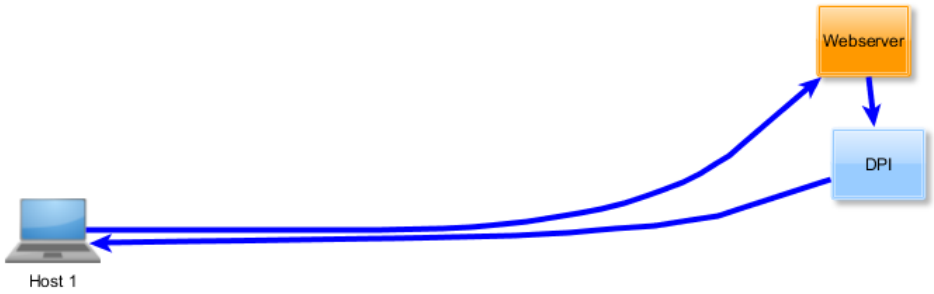


Figure 37: request2

Listing 33: Second service request (service graph) to ESCAPE

```
{
  "parameters": {
    "id": "EWSDN-demo-req2",
    "name": "EWSDN-2web-1dpi-2SAP-req",
    "version": "1.0"
  },
  "node_nfs": [
    {
      "id": "webserver1",
      "name": "webserver1",
      "ports": [
        {
          "id": 0
        }
      ],
      "functional_type": "webserver",
      "specification": {
        "resources": {
          "cpu": 1,
          "mem": 1,
          "storage": 0
        }
      }
    },
    {
      "id": "dpi",
      "name": "DPI",
      "ports": [
        {
          "id": 1
        },
        {
          "id": 2
        }
      ],
      "functional_type": "dpi",
      "specification": {
        "resources": {
          "cpu": 1,
```

```
            "mem": 1,
            "storage": 0
          }
        }
      }
    ],
    "node_saps": [
      {
        "id": "sap1",
        "name": "SAP1",
        "ports": [
          {
            "id": 1
          }
        ]
      },
      {
        "id": "sap2",
        "name": "SAP2",
        "ports": [
          {
            "id": 1
          }
        ]
      }
    ],
    "edge_sg_nexthops": [
      {
        "id": 11,
        "src_node": "sap1",
        "src_port": 1,
        "dst_node": "webserver1",
        "dst_port": 0
      },
      {
        "id": 12,
        "src_node": "webserver1",
        "src_port": 0,
        "dst_node": "dpi",
        "dst_port": 1
      },
      {
        "id": 13,
        "src_node": "dpi",
        "src_port": 2,
        "dst_node": "sap1",
        "dst_port": 1
      }
    ],
    "edge_reqs": [
      {
        "id": 51282000,
        "src_node": "sap1",
        "src_port": 1,
        "dst_node": "sap1",
        "dst_port": 1,
        "delay": 100,
        "bandwidth": 1,
        "sg_path": [
          11,
```

```
            12,
            13
          ]
        }
    ]
}
```

The above service request can be formalized with the Virtualizer model as described in Listing 34.

Listing 34: NFFG of Second request (Virtualizer) to ESCAPE: xml view

```
<?xml version="1.0" ?>
<virtualizer>
  <nodes>
    <node>
      <id>SingleBiSbiS</id>
      <NF_instances>
        <node operation="create">
          <id>dpi</id>
          <name>DPI</name>
          <type>dpi</type>
          <ports>
            <port>
              <id>1</id>
              <port_type>port-abstract</port_type>
            </port>
            <port>
              <id>2</id>
              <port_type>port-abstract</port_type>
            </port>
          </ports>
          <resources>
            <cpu>1.0</cpu>
            <mem>1.0</mem>
            <storage>0.0</storage>
          </resources>
        </node>
      </NF_instances>
      <flowtable>
        <flowentry>
          <id>ESCAPE-flowentry12</id>
          <out operation="replace" >../../../ NF_instances/node[id=dpi]/ports/port[id=1]</out>
        </flowentry>
        <flowentry operation="create">
          <id>ESCAPE-flowentry13</id>
          <name>sg_hop:13</name>
          <priority>100</priority>
          <port >../../../ NF_instances/node[id=dpi]/ports/port[id=2]</port>
          <out >../../../ ports/port[id=port-SAP1]</out>
          <resources>
            <bandwidth>1.0</bandwidth>
          </resources>
        </flowentry>
      </flowtable>
    </node>
  </nodes>
  <version>2016-02-24; compiled at 2016-03-18 19:56:13</version>
</virtualizer>
```

The outcome of the global orchestration is shown in Fig. 38.

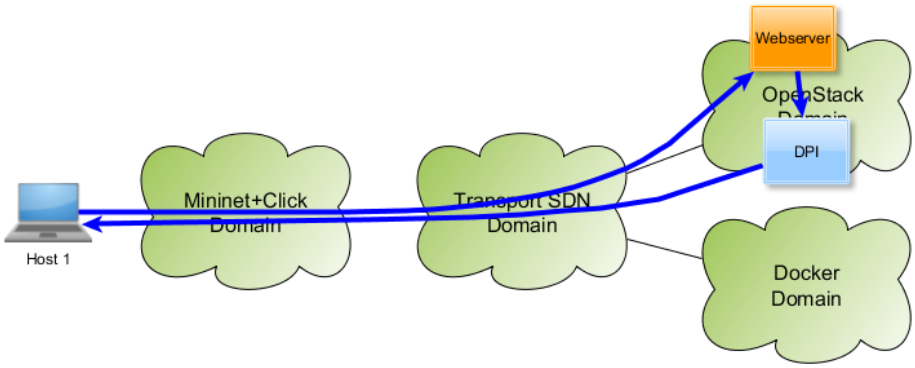


Figure 38: deployed2

The configuration messages sent by the global orchestrator to the OpenStack domain is visible in Listing 35, and to the Mininet domain in Listing 36.

Listing 35: NFFG of Second step configuration message to OpenStack domain: xml view

```
<?xml version="1.0"?>
<virtualizer>
  <nodes>
    <node>
      <id>UUID-OpenStack-01-pack</id>
      <NF_instances>
        <node operation="create">
          <id>dpi</id>
          <name>DPI</name>
          <type>dpi</type>
          <ports>
            <port>
              <id>1</id>
              <port_type>port-abstract</port_type>
            </port>
            <port>
              <id>2</id>
              <port_type>port-abstract</port_type>
            </port>
          </ports>
          <links>
            <link>
              <id>l1</id>
              <name>internal1</name>
              <src>../../../ ports/port[id=1]</src>
              <dst>../../../ ports/port[id=2]</dst>
            </link>
          </links>
          <resources>
            <cpu>1.0</cpu>
            <mem>1.0</mem>
            <storage>0.0</storage>
          </resources>
        </node>
      </NF_instances>
      <flowtable>
        <flowentry>
          <id>ESCAPE-flowentry12</id>
          <action operation="delete">push_tag:0x000c</action>
```

```
40            <out operation="replace">../../../NF_instances/node[id=dpi]/ports/port[id=1]</out>
41          </flowentry>
42          <flowentry operation="create">
43            <id>ESCAPE-flowentry13</id>
44            <name>sg_hop:13</name>
45            <priority>100</priority>
46            <port>../../../NF_instances/node[id=dpi]/ports/port[id=2]</port>
47            <action>push_tag:0x000d</action>
48            <out>../../../ports/port[id=0]</out>
49          </flowentry>
50        </flowtable>
51      </node>
52    </nodes>
53    <version>2016-02-24; compiled at 2016-03-18 19:56:13</version>
54  </virtualizer>
```

Listing 36: NFFG of Second step configuration message to Mininet domain: xml view

```
1  <?xml version="1.0"?>
2  <virtualizer>
3    <nodes>
4      <node>
5        <id>SingleBiSBiS</id>
6        <flowtable>
7          <flowentry operation="delete">
8            <id>ESCAPE-flowentry12</id>
9          </flowentry>
10         <flowentry operation="create">
11           <id>ESCAPE-flowentry13</id>
12           <name>sg_hop:13</name>
13           <priority>100</priority>
14           <port>../../../ports/port[id=port-SAP14]</port>
15           <out>../../../ports/port[id=port-SAP1]</out>
16         </flowentry>
17       </flowtable>
18     </node>
19   </nodes>
20   <version>2016-02-24; compiled at 2016-03-18 19:56:13</version>
21 </virtualizer>
```

The result global configurations configuration is showed in Listing 37, the OpenStack configuration in Listing 38, and the Mininet configuration in Listing 39.

Listing 37: NFFG of Second step result: global configuration: xml view

```
1  <?xml version="1.0"?>
2  <virtualizer>
3    <id>SingleBiSBiS-NFFG</id>
4    <name>Single-BiSBiS-View</name>
5    <nodes>
6      <node>
7        <id>SingleBiSbiS</id>
8        <name>SingleBiSBiS</name>
9        <type>BiSBiS</type>
10       <ports>
11         <port>
12           <id>port-SAP1</id>
13           <name>SAP1</name>
14           <port_type>port-sap</port_type>
15         </port>
```

```
16          <port>
17            <id>port-SAP2</id>
18            <name>SAP2</name>
19            <port_type>port-sap</port_type>
20          </port>
21          <port>
22            <id>port-SAP34</id>
23            <name>SAP34</name>
24            <port_type>port-sap</port_type>
25          </port>
26          <port>
27            <id>port-SAP55</id>
28            <name>SAP55</name>
29            <port_type>port-sap</port_type>
30          </port>
31        </ports>
32        <links>
33          <link>
34            <id>resource-link0</id>
35            <src>../../../ports/port[id=port-SAP34]</src>
36            <dst>../../../ports/port[id=port-SAP55]</dst>
37            <resources>
38              <delay>0</delay>
39              <bandwidth>3840000.0</bandwidth>
40            </resources>
41          </link>
42          <link>
43            <id>resource-link1</id>
44            <src>../../../ports/port[id=port-SAP34]</src>
45            <dst>../../../ports/port[id=port-SAP1]</dst>
46            <resources>
47              <delay>0</delay>
48              <bandwidth>3840000.0</bandwidth>
49            </resources>
50          </link>
51          <link>
52            <id>resource-link2</id>
53            <src>../../../ports/port[id=port-SAP34]</src>
54            <dst>../../../ports/port[id=port-SAP2]</dst>
55            <resources>
56              <delay>0</delay>
57              <bandwidth>3840000.0</bandwidth>
58            </resources>
59          </link>
60          <link>
61            <id>resource-link3</id>
62            <src>../../../ports/port[id=port-SAP55]</src>
63            <dst>../../../ports/port[id=port-SAP1]</dst>
64            <resources>
65              <delay>0</delay>
66              <bandwidth>3840000.0</bandwidth>
67            </resources>
68          </link>
69          <link>
70            <id>resource-link4</id>
71            <src>../../../ports/port[id=port-SAP55]</src>
72            <dst>../../../ports/port[id=port-SAP2]</dst>
73            <resources>
74              <delay>0</delay>
75              <bandwidth>3840000.0</bandwidth>
```

```
            </resources>
          </link>
          <link>
            <id>resource-link5</id>
            <src>../../../ports/port[id=port-SAP1]</src>
            <dst>../../../ports/port[id=port-SAP2]</dst>
            <resources>
              <delay>0</delay>
              <bandwidth>3840000.0</bandwidth>
            </resources>
          </link>
        </links>
        <resources>
          <cpu>29.0</cpu>
          <mem>51082.0</mem>
          <storage>1034.0</storage>
        </resources>
        <metadata>
          <key>bandwidth</key>
          <value>3840000.0</value>
        </metadata>
        <metadata>
          <key>delay</key>
          <value>0</value>
        </metadata>
        <NF_instances>
          <node>
            <id>dpi</id>
            <name>DPI</name>
            <type>dpi</type>
            <ports>
              <port>
                <id>1</id>
                <port_type>port-abstract</port_type>
              </port>
              <port>
                <id>2</id>
                <port_type>port-abstract</port_type>
              </port>
            </ports>
            <resources>
              <cpu>1.0</cpu>
              <mem>1.0</mem>
              <storage>0.0</storage>
            </resources>
          </node>
          <node>
            <id>webserver1</id>
            <name>webserver1</name>
            <type>webserver</type>
            <ports>
              <port>
                <id>0</id>
                <port_type>port-abstract</port_type>
              </port>
            </ports>
            <resources>
              <cpu>1.0</cpu>
              <mem>1.0</mem>
              <storage>0.0</storage>
```

```
          </resources>
        </node>
      </NF_instances>
      < capabilities >
        <supported_NFs>
          <node>
            <id>dpi</id>
            <type>dpi</type>
          </node>
          <node>
            <id>headerCompressor</id>
            <type>headerCompressor</type>
          </node>
          <node>
            <id>headerDecompressor</id>
            <type>headerDecompressor</type>
          </node>
          <node>
            <id> simpleForwarder </id>
            <type> simpleForwarder </type>
          </node>
          <node>
            <id>webserver</id>
            <type>webserver</type>
          </node>
        </supported_NFs>
      </ capabilities >
      <flowtable>
        <flowentry>
          <id>ESCAPE—flowentry11</id>
          <name>sg_hop:11</name>
          < priority >100</ priority >
          <port >../../../ ports/port[id=port—SAP1]</port>
          <out >../../../ NF_instances/node[id=webserver1]/ports/port[id=0]</out>
          <resources>
            <bandwidth>1.0</bandwidth>
          </resources>
        </flowentry>
        <flowentry>
          <id>ESCAPE—flowentry12</id>
          <name>sg_hop:12</name>
          < priority >100</ priority >
          <port >../../../ NF_instances/node[id=webserver1]/ports/port[id=0]</port>
          <out >../../../ NF_instances/node[id=dpi]/ports/port[id=1]</out>
          <resources>
            <bandwidth>1.0</bandwidth>
          </resources>
        </flowentry>
        <flowentry>
          <id>ESCAPE—flowentry13</id>
          <name>sg_hop:13</name>
          < priority >100</ priority >
          <port >../../../ NF_instances/node[id=dpi]/ports/port[id=2]</port>
          <out >../../../ ports/port[id=port—SAP1]</out>
          <resources>
            <bandwidth>1.0</bandwidth>
          </resources>
        </flowentry>
      </flowtable>
    </node>
```

```
196   </nodes>
197   < version >2016-02-24; compiled at  2016-03-18 19:56:13</ version >
198 </ virtualizer >
```

Listing 38: NFFG of Second step result: OpenStack configuration: xml view

```
1  <?xml  version =" 1.0 "  ?>
2  < virtualizer >
3    <id>UUID-ETH-001-pack</id>
4    <name>ETH OpenStack domain-packaged</name>
5    <nodes>
6      <node>
7        <id>UUID-OpenStack-01-pack</id>
8        <name>single  Bis-Bis node  representing  the  whole  OpenStack  domain-packaged</name>
9        <type> BisBis </type>
10       <ports>
11         <port>
12           <id>0</id>
13           <name>SAP24</name>
14           <port_type>port-sap</port_type>
15           <sap>SAP24</sap>
16         </port>
17       </ports>
18       <resources>
19         <cpu>18</cpu>
20         <mem>50944</mem>
21         <storage>1024</storage>
22       </resources>
23       <NF_instances>
24         <node>
25           <id> dpi </id>
26           <name>DPI</name>
27           <type> dpi </type>
28           <ports>
29             <port>
30               <id >1</ id >
31               <port_type>port-abstract</port_type>
32             </port>
33             <port>
34               <id>2</id>
35               <port_type>port-abstract</port_type>
36             </port>
37           </ports>
38           < links >
39             < link >
40               <id> l1 </id>
41               <name> internal1 </name>
42               <src  >../../../  ports/port[ id =1]</ src>
43               <dst  >../../../  ports/port[ id =2]</ dst>
44             </ link >
45           </ links >
46           <resources>
47             <cpu >1.0</ cpu>
48             <mem>1.0</mem>
49             <storage >0.0</ storage>
50           </resources>
51         </node>
52         <node>
53           <id>webserver1</id>
54           <name>webserver1</name>
```

```
        <type>webserver</type>
        <ports>
          <port>
            <id>0</id>
            <port_type>port-abstract</port_type>
          </port>
        </ports>
        <resources>
          <cpu>1.0</cpu>
          <mem>1.0</mem>
          <storage>0.0</storage>
        </resources>
      </node>
    </NF_instances>
    < capabilities >
      <supported_NFs>
        <node>
          <id>dpi</id>
          <name>dpi4</name>
          <type>dpi</type>
          <ports>
            <port>
              <id>1</id>
              <name>input</name>
              <port_type>port-abstract</port_type>
            </port>
            <port>
              <id>2</id>
              <name>output</name>
              <port_type>port-abstract</port_type>
            </port>
          </ports>
          < links >
            < link >
              <id>l1</id>
              <name>internal1</name>
              <src >../../../ ports/port[id=1]</src>
              <dst >../../../ ports/port[id=2]</dst>
            </link>
          </links>
          <resources>
            <cpu>1</cpu>
            <mem>128</mem>
            <storage>1</storage>
          </resources>
        </node>
        <node>
          <id>webserver</id>
          <name>webserver4</name>
          <type>webserver</type>
          <ports>
            <port>
              <id>0</id>
              <name>in-out</name>
              <port_type>port-abstract</port_type>
            </port>
          </ports>
          <resources>
            <cpu>1</cpu>
            <mem>128</mem>
```

```
                <storage>1</storage>
              </resources>
            </node>
          </supported_NFs>
        </ capabilities >
        <flowtable>
          <flowentry>
            <id>ESCAPE-flowentry11</id>
            <name>sg_hop:11</name>
            < priority >100</ priority >
            <port >../../../ ports/port[id=0]</port>
            <match>dl_tag=0x000b</match>
            <action>pop_tag</action>
            <out >../../../ NF_instances/node[id=webserver1]/ports/port[id=0]</out>
            <resources>
              <bandwidth>1.0</bandwidth>
            </resources>
          </flowentry>
          <flowentry>
            <id>ESCAPE-flowentry12</id>
            <name>sg_hop:12</name>
            < priority >100</ priority >
            <port >../../../ NF_instances/node[id=webserver1]/ports/port[id=0]</port>
            <out >../../../ NF_instances/node[id=dpi]/ports/port[id=1]</out>
            <resources>
              <bandwidth>1.0</bandwidth>
            </resources>
          </flowentry>
          <flowentry>
            <id>ESCAPE-flowentry13</id>
            <name>sg_hop:13</name>
            < priority >100</ priority >
            <port >../../../ NF_instances/node[id=dpi]/ports/port[id=2]</port>
            <action>push_tag:0x000d</action>
            <out >../../../ ports/port[id=0]</out>
            <resources>
              <bandwidth>1.0</bandwidth>
            </resources>
          </flowentry>
        </flowtable>
      </node>
    </nodes>
    < version >2016-02-24; compiled at 2016-03-18 19:56:13</ version >
</ virtualizer >
```

Listing 39: NFFG of Second step result: Mininet configuration: xml view

```
<?xml version="1.0" ?>
< virtualizer >
  <id> SingleBiSBiS -NFFG</id>
  <name>Single-BiSBiS-View</name>
  <nodes>
    <node>
      <id> SingleBiSBiS </id>
      <name>SingleBiSBiS</name>
      <type>BiSBiS</type>
      <ports>
        <port>
          <id>port-SAP1</id>
          <name>SAP1</name>
```

```
        <port_type>port-sap</port_type>
      </port>
      <port>
        <id>port-SAP14</id>
        <name>SAP14</name>
        <port_type>port-sap</port_type>
        <sap>SAP14</sap>
      </port>
      <port>
        <id>port-SAP2</id>
        <name>SAP2</name>
        <port_type>port-sap</port_type>
      </port>
    </ports>
    <links>
      <link>
        <id>resource-link0</id>
        <src>../../../ports/port[id=port-SAP14]</src>
        <dst>../../../ports/port[id=port-SAP1]</dst>
        <resources>
          <delay>0.2</delay>
          <bandwidth>160000.0</bandwidth>
        </resources>
      </link>
      <link>
        <id>resource-link1</id>
        <src>../../../ports/port[id=port-SAP14]</src>
        <dst>../../../ports/port[id=port-SAP2]</dst>
        <resources>
          <delay>0.2</delay>
          <bandwidth>160000.0</bandwidth>
        </resources>
      </link>
      <link>
        <id>resource-link2</id>
        <src>../../../ports/port[id=port-SAP1]</src>
        <dst>../../../ports/port[id=port-SAP2]</dst>
        <resources>
          <delay>0.2</delay>
          <bandwidth>160000.0</bandwidth>
        </resources>
      </link>
    </links>
    <resources>
      <cpu>10.0</cpu>
      <mem>10.0</mem>
      <storage>10.0</storage>
    </resources>
    <metadata>
      <key>bandwidth</key>
      <value>160000.0</value>
    </metadata>
    <metadata>
      <key>delay</key>
      <value>0.2</value>
    </metadata>
    <capabilities>
      <supported_NFs>
        <node>
          <id>headerCompressor</id>
```

```
            <type>headerCompressor</type>
          </node>
          <node>
            <id>headerDecompressor</id>
            <type>headerDecompressor</type>
          </node>
          <node>
            <id> simpleForwarder </id>
            <type> simpleForwarder </type>
          </node>
        </supported_NFs>
      </ capabilities >
      <flowtable>
        <flowentry>
          <id>ESCAPE-flowentry11</id>
          <name>sg_hop:11</name>
          < priority >100</ priority >
          <port >../../../ ports/port[id=port-SAP1]</port>
          <out >../../../ ports/port[id=port-SAP14]</out>
          <resources>
            <bandwidth>0.0</bandwidth>
          </resources>
        </flowentry>
        <flowentry>
          <id>ESCAPE-flowentry13</id>
          <name>sg_hop:13</name>
          < priority >100</ priority >
          <port >../../../ ports/port[id=port-SAP14]</port>
          <out >../../../ ports/port[id=port-SAP1]</out>
          <resources>
            <bandwidth>1.0</bandwidth>
          </resources>
        </flowentry>
      </flowtable>
    </node>
  </nodes>
  < version >2016-02-24; compiled at 2016-03-18 19:56:13</ version >
</ virtualizer >
```

A.5 Service Request 3: TCP header (de)compressor added

In the third example we extend the existing setup with requesting a TCP header compressor and a TCP header decompressor. The third service request is depicted in Fig. 39, and formalized in Listing 40 to be sent to ESCAPE.

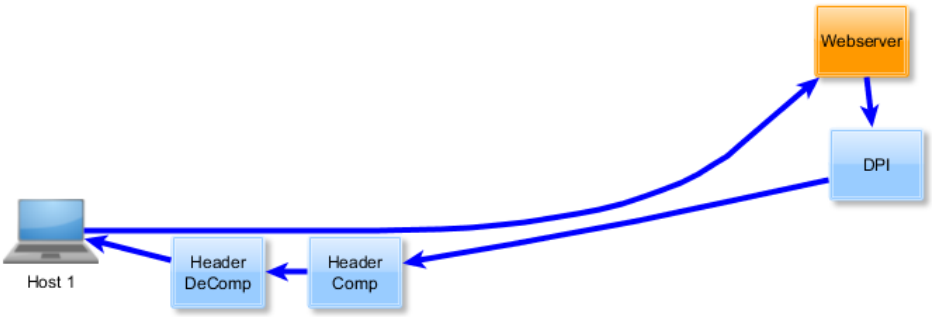


Figure 39: request3

Listing 40: Third service request (service graph) to ESCAPE

```
{
  "parameters": {
    "id": "EWSDN-demo-req3",
    "name": "EWSDN-2web-1dpi-1comp-1decomp-2SAP-req",
    "version": "1.0"
  },
  "node_nfs": [
    {
      "id": "webserver1",
      "name": "webserver1",
      "ports": [
        {
          "id": 0
        }
      ],
      "functional_type": "webserver",
      "specification": {
        "resources": {
          "cpu": 1,
          "mem": 1,
          "storage": 0
        }
      }
    },
    {
      "id": "dpi",
      "name": "DPI",
      "ports": [
        {
          "id": 1
        },
        {
          "id": 2
        }
      ],
      "functional_type": "dpi",
      "specification": {
        "resources": {
```

```
            "cpu": 1,
            "mem": 1,
            "storage": 0
          }
        }
      },
      {
        "id": "comp",
        "name": "COMPRESSOR",
        "ports": [
          {
            "id": 1
          }
        ],
        "functional_type": "headerCompressor",
        "specification": {
          "resources": {
            "cpu": 1,
            "mem": 1,
            "storage": 0
          }
        }
      },
      {
        "id": "decomp",
        "name": "DECOMPRESSOR",
        "ports": [
          {
            "id": 1
          }
        ],
        "functional_type": "headerDecompressor",
        "specification": {
          "resources": {
            "cpu": 1,
            "mem": 1,
            "storage": 0
          }
        }
      }
    ],
    "node_saps": [
      {
        "id": "sap2",
        "name": "SAP2",
        "ports": [
          {
            "id": 1
          }
        ]
      },
      {
        "id": "sap1",
        "name": "SAP1",
        "ports": [
          {
            "id": 1
          }
        ]
      }
```

```
  ],
  "edge_sg_nexthops": [
    {
      "id": 11,
      "src_node": "sap1",
      "src_port": 1,
      "dst_node": "webserver1",
      "dst_port": 0
    },
    {
      "id": 12,
      "src_node": "webserver1",
      "src_port": 0,
      "dst_node": "dpi",
      "dst_port": 1
    },
    {
      "id": 13,
      "src_node": "dpi",
      "src_port": 2,
      "dst_node": "comp",
      "dst_port": 1
    },
    {
      "id": 14,
      "src_node": "comp",
      "src_port": 1,
      "dst_node": "decomp",
      "dst_port": 1
    },
    {
      "id": 15,
      "src_node": "decomp",
      "src_port": 1,
      "dst_node": "sap1",
      "dst_port": 1
    }
  ],
  "edge_reqs": [
    {
      "id": 36246224,
      "src_node": "sap1",
      "src_port": 1,
      "dst_node": "sap1",
      "dst_port": 1,
      "delay": 100,
      "bandwidth": 1,
      "sg_path": [
        11,
        12,
        13,
        14,
        15
      ]
    }
  ]
}
```


The above service request can be formalized with the Virtualizer model as described in Listing 41.

Listing 41: NFFG of Third request (Virtualizer) to ESCAPE: xml view

```
<?xml version="1.0" ?>
<virtualizer>
  <nodes>
    <node>
      <id>SingleBiSbiS</id>
      <NF_instances>
        <node operation="create">
          <id>comp</id>
          <name>COMPRESSOR</name>
          <type>headerCompressor</type>
          <ports>
            <port>
              <id>1</id>
              <port_type>port-abstract</port_type>
            </port>
          </ports>
          <resources>
            <cpu>1.0</cpu>
            <mem>1.0</mem>
            <storage>0.0</storage>
          </resources>
        </node>
        <node operation="create">
          <id>decomp</id>
          <name>DECOMPRESSOR</name>
          <type>headerDecompressor</type>
          <ports>
            <port>
              <id>1</id>
              <port_type>port-abstract</port_type>
            </port>
          </ports>
          <resources>
            <cpu>1.0</cpu>
            <mem>1.0</mem>
            <storage>0.0</storage>
          </resources>
        </node>
      </NF_instances>
      <flowtable>
        <flowentry>
          <id>ESCAPE-flowentry13</id>
          <out operation="replace">../../../NF_instances/node[id=comp]/ports/port[id=1]</out>
        </flowentry>
        <flowentry operation="create">
          <id>ESCAPE-flowentry14</id>
          <name>sg_hop:14</name>
          <priority>100</priority>
          <port>../../../NF_instances/node[id=comp]/ports/port[id=1]</port>
          <out>../../../NF_instances/node[id=decomp]/ports/port[id=1]</out>
          <resources>
            <bandwidth>1.0</bandwidth>
          </resources>
        </flowentry>
        <flowentry operation="create">
          <id>ESCAPE-flowentry15</id>
          <name>sg_hop:15</name>
          <priority>100</priority>
          <port>../../../NF_instances/node[id=decomp]/ports/port[id=1]</port>
```

```
            <out >../../../ ports/port[id=port-SAP1]</out>
            <resources>
              <bandwidth>1.0</bandwidth>
            </resources>
          </flowentry>
        </flowtable>
      </node>
    </nodes>
    < version >2016-02-24; compiled at 2016-03-18 19:56:13</ version >
</ virtualizer >
```

The outcome of the global orchestration is shown in Fig. 40.

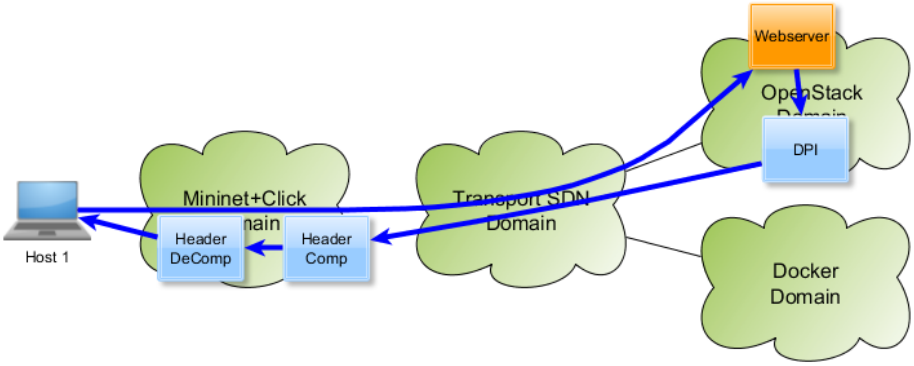


Figure 40: deployed3

The configuration messages sent by the global orchestrator to the OpenStack domain is visible in Listing 42, and to the Mininet domain in Listing 43.

Listing 42: NFFG of Third step configuration message to OpenStack domain: xml view

```
<?xml version = " 1.0 " ?>
< virtualizer >
  < version >2016-02-24; compiled at 2016-03-18 19:56:13</ version >
</ virtualizer >
```

Listing 43: NFFG of Third step configuration message to Mininet domain: xml view

```
<?xml version = " 1.0 " ?>
< virtualizer >
  <nodes>
    <node>
      <id> SingleBiSBiS </id>
      <NF_instances>
        <node operation = " create ">
          <id>comp</id>
          <name>COMPRESSOR</name>
          <type>headerCompressor</type>
          <ports>
            <port>
              <id>1</id>
              <port_type>port-abstract</port_type>
            </port>
          </ports>
          <resources>
            <cpu>1.0</cpu>
```

```
            <mem>1.0</mem>
            <storage>0.0</storage>
          </resources>
        </node>
        <node operation="create">
          <id>decomp</id>
          <name>DECOMPRESSOR</name>
          <type>headerDecompressor</type>
          <ports>
            <port>
              <id>1</id>
              <port_type>port-abstract</port_type>
            </port>
          </ports>
          <resources>
            <cpu>1.0</cpu>
            <mem>1.0</mem>
            <storage>0.0</storage>
          </resources>
        </node>
      </NF_instances>
      <flowtable>
        <flowentry>
          <id>ESCAPE-flowentry13</id>
          <out operation="replace">../../../NF_instances/node[id=comp]/ports/port[id=1]</out>
        </flowentry>
        <flowentry operation="create">
          <id>ESCAPE-flowentry14</id>
          <name>sg_hop:14</name>
          <priority>100</priority>
          <port>../../../NF_instances/node[id=comp]/ports/port[id=1]</port>
          <out>../../../NF_instances/node[id=decomp]/ports/port[id=1]</out>
        </flowentry>
        <flowentry operation="create">
          <id>ESCAPE-flowentry15</id>
          <name>sg_hop:15</name>
          <priority>100</priority>
          <port>../../../NF_instances/node[id=decomp]/ports/port[id=1]</port>
          <out>../../../ports/port[id=port-SAP1]</out>
        </flowentry>
      </flowtable>
    </node>
  </nodes>
  <version>2016-02-24; compiled at 2016-03-18 19:56:13</version>
</virtualizer>
```

The result global configurations configuration is showed in Listing 44, the OpenStack configuration in Listing 45, and the Mininet configuration in Listing 46.

Listing 44: NFFG of Third step result: global configuration: xml view

```
<?xml version="1.0"?>
<virtualizer>
  <id>SingleBiSBiS-NFFG</id>
  <name>Single-BiSBiS-View</name>
  <nodes>
    <node>
      <id>SingleBiSbiS</id>
      <name>SingleBiSBiS</name>
      <type>BiSBiS</type>
      <ports>
```

```
        <port>
          <id>port-SAP1</id>
          <name>SAP1</name>
          <port_type>port-sap</port_type>
        </port>
        <port>
          <id>port-SAP2</id>
          <name>SAP2</name>
          <port_type>port-sap</port_type>
        </port>
        <port>
          <id>port-SAP34</id>
          <name>SAP34</name>
          <port_type>port-sap</port_type>
        </port>
        <port>
          <id>port-SAP55</id>
          <name>SAP55</name>
          <port_type>port-sap</port_type>
        </port>
      </ports>
      <links>
        <link>
          <id>resource-link0</id>
          <src>../../../ports/port[id=port-SAP34]</src>
          <dst>../../../ports/port[id=port-SAP55]</dst>
          <resources>
            <delay>0</delay>
            <bandwidth>4480000.0</bandwidth>
          </resources>
        </link>
        <link>
          <id>resource-link1</id>
          <src>../../../ports/port[id=port-SAP34]</src>
          <dst>../../../ports/port[id=port-SAP2]</dst>
          <resources>
            <delay>0</delay>
            <bandwidth>4480000.0</bandwidth>
          </resources>
        </link>
        <link>
          <id>resource-link2</id>
          <src>../../../ports/port[id=port-SAP34]</src>
          <dst>../../../ports/port[id=port-SAP1]</dst>
          <resources>
            <delay>0</delay>
            <bandwidth>4480000.0</bandwidth>
          </resources>
        </link>
        <link>
          <id>resource-link3</id>
          <src>../../../ports/port[id=port-SAP55]</src>
          <dst>../../../ports/port[id=port-SAP2]</dst>
          <resources>
            <delay>0</delay>
            <bandwidth>4480000.0</bandwidth>
          </resources>
        </link>
        <link>
          <id>resource-link4</id>
```

```
            <src >../../../ ports/port[id=port-SAP55]</src>
            <dst >../../../ ports/port[id=port-SAP1]</dst>
            <resources>
              <delay>0</delay>
              <bandwidth>4480000.0</bandwidth>
            </resources>
          </link>
          <link>
            <id>resource-link5</id>
            <src >../../../ ports/port[id=port-SAP2]</src>
            <dst >../../../ ports/port[id=port-SAP1]</dst>
            <resources>
              <delay>0</delay>
              <bandwidth>4480000.0</bandwidth>
            </resources>
          </link>
        </links>
        <resources>
          <cpu>29.0</cpu>
          <mem>51082.0</mem>
          <storage>1034.0</storage>
        </resources>
        <metadata>
          <key>bandwidth</key>
          <value>4480000.0</value>
        </metadata>
        <metadata>
          <key>delay</key>
          <value>0</value>
        </metadata>
        <NF_instances>
          <node>
            <id>comp</id>
            <name>COMPRESSOR</name>
            <type>headerCompressor</type>
            <ports>
              <port>
                <id>1</id>
                <port_type>port-abstract</port_type>
              </port>
            </ports>
            <resources>
              <cpu>1.0</cpu>
              <mem>1.0</mem>
              <storage>0.0</storage>
            </resources>
          </node>
          <node>
            <id>decomp</id>
            <name>DECOMPRESSOR</name>
            <type>headerDecompressor</type>
            <ports>
              <port>
                <id>1</id>
                <port_type>port-abstract</port_type>
              </port>
            </ports>
            <resources>
              <cpu>1.0</cpu>
              <mem>1.0</mem>
```

```
            <storage>0.0</storage>
          </resources>
        </node>
        <node>
          <id>dpi</id>
          <name>DPI</name>
          <type>dpi</type>
          <ports>
            <port>
              <id>1</id>
              <port_type>port-abstract</port_type>
            </port>
            <port>
              <id>2</id>
              <port_type>port-abstract</port_type>
            </port>
          </ports>
          <resources>
            <cpu>1.0</cpu>
            <mem>1.0</mem>
            <storage>0.0</storage>
          </resources>
        </node>
        <node>
          <id>webserver1</id>
          <name>webserver1</name>
          <type>webserver</type>
          <ports>
            <port>
              <id>0</id>
              <port_type>port-abstract</port_type>
            </port>
          </ports>
          <resources>
            <cpu>1.0</cpu>
            <mem>1.0</mem>
            <storage>0.0</storage>
          </resources>
        </node>
      </NF_instances>
      < capabilities >
        <supported_NFs>
          <node>
            <id>dpi</id>
            <type>dpi</type>
          </node>
          <node>
            <id>headerCompressor</id>
            <type>headerCompressor</type>
          </node>
          <node>
            <id>headerDecompressor</id>
            <type>headerDecompressor</type>
          </node>
          <node>
            <id> simpleForwarder </id>
            <type> simpleForwarder </type>
          </node>
          <node>
            <id>webserver</id>
```

```
            <type>webserver</type>
          </node>
        </supported_NFs>
      </ capabilities >
      <flowtable>
        <flowentry>
          <id>ESCAPE-flowentry11</id>
          <name>sg_hop:11</name>
          < priority >100</ priority >
          <port >../../../ ports/port[id=port-SAP1]</port>
          <out >../../../ NF_instances/node[id=webserver1]/ports/port[id=0]</out>
          <resources>
            <bandwidth>1.0</bandwidth>
          </resources>
        </flowentry>
        <flowentry>
          <id>ESCAPE-flowentry12</id>
          <name>sg_hop:12</name>
          < priority >100</ priority >
          <port >../../../ NF_instances/node[id=webserver1]/ports/port[id=0]</port>
          <out >../../../ NF_instances/node[id=dpi]/ports/port[id=1]</out>
          <resources>
            <bandwidth>1.0</bandwidth>
          </resources>
        </flowentry>
        <flowentry>
          <id>ESCAPE-flowentry13</id>
          <name>sg_hop:13</name>
          < priority >100</ priority >
          <port >../../../ NF_instances/node[id=dpi]/ports/port[id=2]</port>
          <out >../../../ NF_instances/node[id=comp]/ports/port[id=1]</out>
          <resources>
            <bandwidth>1.0</bandwidth>
          </resources>
        </flowentry>
        <flowentry>
          <id>ESCAPE-flowentry14</id>
          <name>sg_hop:14</name>
          < priority >100</ priority >
          <port >../../../ NF_instances/node[id=comp]/ports/port[id=1]</port>
          <out >../../../ NF_instances/node[id=decomp]/ports/port[id=1]</out>
          <resources>
            <bandwidth>1.0</bandwidth>
          </resources>
        </flowentry>
        <flowentry>
          <id>ESCAPE-flowentry15</id>
          <name>sg_hop:15</name>
          < priority >100</ priority >
          <port >../../../ NF_instances/node[id=decomp]/ports/port[id=1]</port>
          <out >../../../ ports/port[id=port-SAP1]</out>
          <resources>
            <bandwidth>1.0</bandwidth>
          </resources>
        </flowentry>
      </flowtable>
    </node>
  </nodes>
  < version >2016-02-24; compiled at 2016-03-18 19:56:13</ version >
</ virtualizer >
```

Listing 45: NFFG of Third step result: OpenStack configuration: xml view

```
<?xml version="1.0"?>
<virtualizer>
  <id>UUID-ETH-001-pack</id>
  <name>ETH OpenStack domain-packaged</name>
  <nodes>
    <node>
      <id>UUID-OpenStack-01-pack</id>
      <name>single Bis-Bis node representing the whole OpenStack domain-packaged</name>
      <type>BisBis</type>
      <ports>
        <port>
          <id>0</id>
          <name>SAP24</name>
          <port_type>port-sap</port_type>
          <sap>SAP24</sap>
        </port>
      </ports>
      <resources>
        <cpu>18</cpu>
        <mem>50944</mem>
        <storage>1024</storage>
      </resources>
      <NF_instances>
        <node>
          <id>dpi</id>
          <name>DPI</name>
          <type>dpi</type>
          <ports>
            <port>
              <id>1</id>
              <port_type>port-abstract</port_type>
            </port>
            <port>
              <id>2</id>
              <port_type>port-abstract</port_type>
            </port>
          </ports>
          <links>
            <link>
              <id>l1</id>
              <name>internal1</name>
              <src>../../../ports/port[id=1]</src>
              <dst>../../../ports/port[id=2]</dst>
            </link>
          </links>
          <resources>
            <cpu>1.0</cpu>
            <mem>1.0</mem>
            <storage>0.0</storage>
          </resources>
        </node>
        <node>
          <id>webserver1</id>
          <name>webserver1</name>
          <type>webserver</type>
          <ports>
            <port>
              <id>0</id>
              <port_type>port-abstract</port_type>
```

```
            </port>
          </ports>
          <resources>
            <cpu>1.0</cpu>
            <mem>1.0</mem>
            <storage>0.0</storage>
          </resources>
        </node>
      </NF_instances>
      < capabilities >
        <supported_NFs>
          <node>
            <id>dpi</id>
            <name>dpi4</name>
            <type>dpi</type>
            <ports>
              <port>
                <id>1</id>
                <name>input</name>
                <port_type>port-abstract</port_type>
              </port>
              <port>
                <id>2</id>
                <name>output</name>
                <port_type>port-abstract</port_type>
              </port>
            </ports>
            < links >
              < link >
                <id>l1</id>
                <name>internal1</name>
                <src >../../../ ports/port[id=1]</src>
                <dst >../../../ ports/port[id=2]</dst>
              </ link >
            </ links >
            <resources>
              <cpu>1</cpu>
              <mem>128</mem>
              <storage>1</storage>
            </resources>
          </node>
          <node>
            <id>webserver</id>
            <name>webserver4</name>
            <type>webserver</type>
            <ports>
              <port>
                <id>0</id>
                <name>in-out</name>
                <port_type>port-abstract</port_type>
              </port>
            </ports>
            <resources>
              <cpu>1</cpu>
              <mem>128</mem>
              <storage>1</storage>
            </resources>
          </node>
        </supported_NFs>
      </ capabilities >
```

```
        <flowtable>
          <flowentry>
            <id>ESCAPE-flowentry11</id>
            <name>sg_hop:11</name>
            < priority >100</ priority >
            <port >../../../ ports/port[id=0]</port>
            <match>dl_tag=0x000b</match>
            <action>pop_tag</action>
            <out >../../../ NF_instances/node[id=webserver1]/ports/port[id=0]</out>
            <resources>
              <bandwidth>1.0</bandwidth>
            </resources>
          </flowentry>
          <flowentry>
            <id>ESCAPE-flowentry12</id>
            <name>sg_hop:12</name>
            < priority >100</ priority >
            <port >../../../ NF_instances/node[id=webserver1]/ports/port[id=0]</port>
            <out >../../../ NF_instances/node[id=dpi]/ports/port[id=1]</out>
            <resources>
              <bandwidth>1.0</bandwidth>
            </resources>
          </flowentry>
          <flowentry>
            <id>ESCAPE-flowentry13</id>
            <name>sg_hop:13</name>
            < priority >100</ priority >
            <port >../../../ NF_instances/node[id=dpi]/ports/port[id=2]</port>
            <action>push_tag:0x000d</action>
            <out >../../../ ports/port[id=0]</out>
            <resources>
              <bandwidth>1.0</bandwidth>
            </resources>
          </flowentry>
        </flowtable>
      </node>
    </nodes>
    < version >2016-02-24; compiled at 2016-03-18 19:56:13</ version >
</ virtualizer >
```

Listing 46: NFFG of Third step result: Mininet configuration: xml view

```
<?xml version="1.0" ?>
< virtualizer >
  <id> SingleBiSBiS -NFFG</id>
  <name>Single-BiSBiS-View</name>
  <nodes>
    <node>
      <id> SingleBiSBiS </id>
      <name> SingleBiSBiS </name>
      <type> BiSBiS </type>
      <ports>
        <port>
          <id>port-SAP1</id>
          <name>SAP1</name>
          <port_type>port-sap</port_type>
        </port>
        <port>
          <id>port-SAP14</id>
          <name>SAP14</name>
```

```
        <port_type>port-sap</port_type>
        <sap>SAP14</sap>
      </port>
      <port>
        <id>port-SAP2</id>
        <name>SAP2</name>
        <port_type>port-sap</port_type>
      </port>
    </ports>
    <links>
      <link>
        <id>resource-link0</id>
        <src>../../../ports/port[id=port-SAP2]</src>
        <dst>../../../ports/port[id=port-SAP14]</dst>
        <resources>
          <delay>0.2</delay>
          <bandwidth>200000.0</bandwidth>
        </resources>
      </link>
      <link>
        <id>resource-link1</id>
        <src>../../../ports/port[id=port-SAP2]</src>
        <dst>../../../ports/port[id=port-SAP1]</dst>
        <resources>
          <delay>0.2</delay>
          <bandwidth>200000.0</bandwidth>
        </resources>
      </link>
      <link>
        <id>resource-link2</id>
        <src>../../../ports/port[id=port-SAP14]</src>
        <dst>../../../ports/port[id=port-SAP1]</dst>
        <resources>
          <delay>0.2</delay>
          <bandwidth>200000.0</bandwidth>
        </resources>
      </link>
    </links>
    <resources>
      <cpu>10.0</cpu>
      <mem>10.0</mem>
      <storage>10.0</storage>
    </resources>
    <metadata>
      <key>bandwidth</key>
      <value>200000.0</value>
    </metadata>
    <metadata>
      <key>delay</key>
      <value>0.2</value>
    </metadata>
    <NF_instances>
      <node>
        <id>comp</id>
        <name>COMPRESSOR</name>
        <type>headerCompressor</type>
        <ports>
          <port>
            <id>1</id>
            <port_type>port-abstract</port_type>
```

```
            </port>
          </ports>
          <resources>
            <cpu>1.0</cpu>
            <mem>1.0</mem>
            <storage>0.0</storage>
          </resources>
        </node>
        <node>
          <id>decomp</id>
          <name>DECOMPRESSOR</name>
          <type>headerDecompressor</type>
          <ports>
            <port>
              <id>1</id>
              <port_type>port-abstract</port_type>
            </port>
          </ports>
          <resources>
            <cpu>1.0</cpu>
            <mem>1.0</mem>
            <storage>0.0</storage>
          </resources>
        </node>
      </NF_instances>
      < capabilities >
        <supported_NFs>
          <node>
            <id>headerCompressor</id>
            <type>headerCompressor</type>
          </node>
          <node>
            <id>headerDecompressor</id>
            <type>headerDecompressor</type>
          </node>
          <node>
            <id> simpleForwarder </id>
            <type> simpleForwarder </type>
          </node>
        </supported_NFs>
      </ capabilities >
      <flowtable>
        <flowentry>
          <id>ESCAPE-flowentry11</id>
          <name>sg_hop:11</name>
          < priority >100</ priority >
          <port >../../../ ports/port[id=port-SAP1]</port>
          <out >../../../ ports/port[id=port-SAP14]</out>
          <resources>
            <bandwidth>0.0</bandwidth>
          </resources>
        </flowentry>
        <flowentry>
          <id>ESCAPE-flowentry13</id>
          <name>sg_hop:13</name>
          < priority >100</ priority >
          <port >../../../ ports/port[id=port-SAP14]</port>
          <out >../../../ NF_instances/node[id=comp]/ports/port[id =1]</out>
          <resources>
            <bandwidth>1.0</bandwidth>
```

```
            </resources>
          </flowentry>
          <flowentry>
            <id>ESCAPE-flowentry14</id>
            <name>sg_hop:14</name>
            < priority >100</ priority >
            <port >../../../ NF_instances/node[id=comp]/ports/port[id =1]</ port>
            <out >../../../ NF_instances/node[id=decomp]/ports/port[id =1]</ out>
            <resources>
              <bandwidth>1.0</bandwidth>
            </resources>
          </flowentry>
          <flowentry>
            <id>ESCAPE-flowentry15</id>
            <name>sg_hop:15</name>
            < priority >100</ priority >
            <port >../../../ NF_instances/node[id=decomp]/ports/port[id =1]</ port>
            <out >../../../ ports/port[id=port-SAP1]</out>
            <resources>
              <bandwidth>1.0</bandwidth>
            </resources>
          </flowentry>
        </ flowtable >
      </node>
    </nodes>
    < version >2016-02-24; compiled at 2016-03-18 19:56:13</ version >
</ virtualizer >
```

A.6 Service Request 4: Ethernet bridge added

In the fourth example we extend the existing setup with an Ethernet bridge NF between the DPI and the TCP header compressor. The fourth service request is depicted in Fig. 41, and formalized in Listing 47 to be sent to ESCAPE.

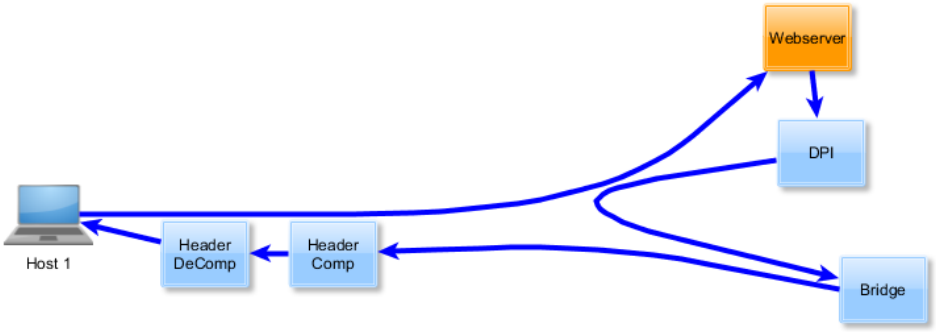


Figure 41: request4

Listing 47: Fourth service request (service graph) to ESCAPE

```
{
  "parameters": {
    "id": "EWSDN-demo-req3",
    "name": "EWSDN-2web-1dpi-1comp-1decomp-2SAP-req",
    "version": "1.0"
  },
  "node_nfs": [
    {
      "id": "webserver1",
      "name": "webserver1",
      "ports": [
        {
          "id": 0
        }
      ],
      "functional_type": "webserver",
      "specification": {
        "resources": {
          "cpu": 1,
          "mem": 1,
          "storage": 0
        }
      }
    },
    {
      "id": "dpi",
      "name": "DPI",
      "ports": [
        {
          "id": 1
        },
        {
          "id": 2
        }
      ],
      "functional_type": "dpi",
      "specification": {
        "resources": {
```

```
            "cpu": 1,
            "mem": 1,
            "storage": 0
          }
        }
      },
      {
        "id": "comp",
        "name": "COMPRESSOR",
        "ports": [
          {
            "id": 1
          }
        ],
        "functional_type": "headerCompressor",
        "specification": {
          "resources": {
            "cpu": 1,
            "mem": 1,
            "storage": 0
          }
        }
      },
      {
        "id": "decomp",
        "name": "DECOMPRESSOR",
        "ports": [
          {
            "id": 1
          }
        ],
        "functional_type": "headerDecompressor",
        "specification": {
          "resources": {
            "cpu": 1,
            "mem": 1,
            "storage": 0
          }
        }
      },
      {
        "id": "dockernf",
        "name": "dockernf",
        "ports": [
          {
            "id": 1
          },
          {
            "id": 2
          }
        ],
        "functional_type": "bridge",
        "specification": {
          "resources": {
            "cpu": 1,
            "mem": 1,
            "storage": 0
          }
        }
      }
```

```
  ],
  "node_saps": [
    {
      "id": "sap2",
      "name": "SAP2",
      "ports": [
        {
          "id": 1
        }
      ]
    },
    {
      "id": "sap1",
      "name": "SAP1",
      "ports": [
        {
          "id": 1
        }
      ]
    }
  ],
  "edge_sg_nexthops": [
    {
      "id": 11,
      "src_node": "sap1",
      "src_port": 1,
      "dst_node": "webserver1",
      "dst_port": 0
    },
    {
      "id": 12,
      "src_node": "webserver1",
      "src_port": 0,
      "dst_node": "dpi",
      "dst_port": 1
    },
    {
      "id": 13,
      "src_node": "dpi",
      "src_port": 2,
      "dst_node": "dockernf",
      "dst_port": 1
    },
    {
      "id": 14,
      "src_node": "dockernf",
      "src_port": 2,
      "dst_node": "comp",
      "dst_port": 1
    },

    {
      "id": 15,
      "src_node": "comp",
      "src_port": 1,
      "dst_node": "decomp",
      "dst_port": 1
    },
    {
      "id": 16,
```

```
        "src_node": "decomp",
        "src_port": 1,
        "dst_node": "sap1",
        "dst_port": 1
      }
    ],
    "edge_reqs": [
      {
        "id": 36246224,
        "src_node": "sap1",
        "src_port": 1,
        "dst_node": "sap1",
        "dst_port": 1,
        "delay": 100,
        "bandwidth": 1,
        "sg_path": [
          11,
          12,
          13,
          14,
          15,
          16
        ]
      }
    ]
}
```


The above service request can be formalized with the Virtualizer model as described in Listing 48.

Listing 48: NFFG of Fourth request (Virtualizer) to ESCAPE: xml view


```
<?xml version="1.0" ?>
<virtualizer>
  <nodes>
    <node>
      <id>SingleBiSBiS</id>
      <NF_instances>
        <node operation="create">
          <id>dockernf</id>
          <name>dockernf</name>
          <type>bridge</type>
          <ports>
            <port>
              <id>1</id>
              <port_type>port-abstract</port_type>
            </port>
            <port>
              <id>2</id>
              <port_type>port-abstract</port_type>
            </port>
          </ports>
          <resources>
            <cpu>1.0</cpu>
            <mem>1.0</mem>
            <storage>0.0</storage>
          </resources>
        </node>
      </NF_instances>
      <flowtable>
        <flowentry>
          <id>ESCAPE-flowentry13</id>
```

```
          <out operation="replace" >../../../ NF_instances/node[id=dockernf]/ports/port[id=1]</out>
        </flowentry>
        <flowentry>
          <id>ESCAPE-flowentry14</id>
          <port operation="replace" >../../../ NF_instances/node[id=dockernf]/ports/port[id=2]</port>
          <out operation="replace" >../../../ NF_instances/node[id=comp]/ports/port[id=1]</out>
        </flowentry>
        <flowentry>
          <id>ESCAPE-flowentry15</id>
          <port operation="replace" >../../../ NF_instances/node[id=comp]/ports/port[id=1]</port>
          <out operation="replace" >../../../ NF_instances/node[id=decomp]/ports/port[id=1]</out>
        </flowentry>
        <flowentry operation="create">
          <id>ESCAPE-flowentry16</id>
          <name>sg_hop:16</name>
          < priority >100</ priority >
          <port >../../../ NF_instances/node[id=decomp]/ports/port[id=1]</port>
          <out >../../../ ports/port[id=port-SAP1]</out>
        </flowentry>
      </flowtable>
    </node>
  </nodes>
  < version >2016-02-24; compiled at 2016-03-18 19:56:13</ version >
</ virtualizer >
```

The outcome of the global orchestration is shown in Fig. 42.

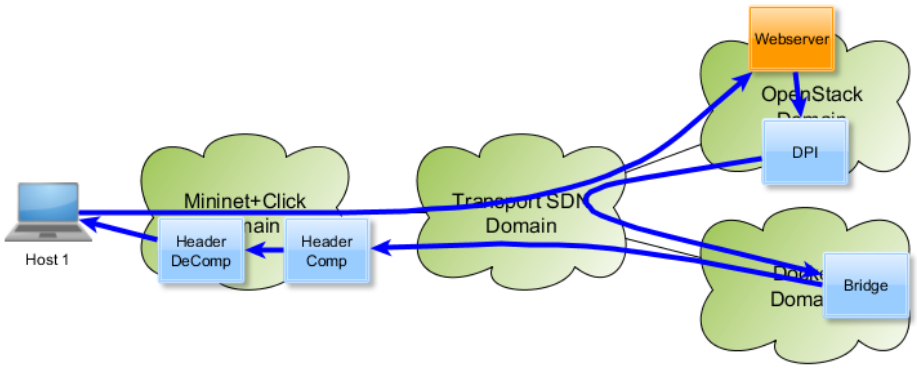


Figure 42: deployed4

The configuration messages sent by the global orchestrator to the Docker domain is visible in Listing 49, and to the Mininet domain in Listing 50.

Listing 49: NFFG of Third step configuration message to Docker domain: xml view

```
<?xml version="1.0" ?>
< virtualizer >
  <nodes>
    <node>
      <id>DOCKER_HOST</id>
      <NF_instances>
        <node operation="create">
          <id>dockernf</id>
          <name>dockernf</name>
          <type>bridge</type>
          <ports>
            <port>
```

```
                <id>1</id>
                <port_type>port-abstract</port_type>
              </port>
              <port>
                <id>2</id>
                <port_type>port-abstract</port_type>
              </port>
            </ports>
            <resources>
              <cpu>1.0</cpu>
              <mem>1.0</mem>
              <storage>0.0</storage>
            </resources>
          </node>
        </NF_instances>
        <flowtable>
          <flowentry operation="create">
            <id>ESCAPE-flowentry13</id>
            <name>sg_hop:13</name>
            <priority>100</priority>
            <port>../../../ports/port[id=eth0]</port>
            <match>dl_tag=0x000d</match>
            <action>pop_tag</action>
            <out>../../../NF_instances/node[id=dockernf]/ports/port[id=1]</out>
          </flowentry>
          <flowentry operation="create">
            <id>ESCAPE-flowentry14</id>
            <name>sg_hop:14</name>
            <priority>100</priority>
            <port>../../../NF_instances/node[id=dockernf]/ports/port[id=2]</port>
            <action>push_tag:0x000e</action>
            <out>../../../ports/port[id=eth0]</out>
          </flowentry>
        </flowtable>
      </node>
    </nodes>
    <version>2016-02-24; compiled at 2016-03-18 19:56:13</version>
</virtualizer>
```

Listing 50: NFFG of Third step configuration message to Mininet domain: xml view

```
<?xml version="1.0"?>
<virtualizer>
  <nodes>
    <node>
      <id>SingleBiSBiS</id>
      <flowtable>
        <flowentry operation="delete">
          <id>ESCAPE-flowentry13</id>
        </flowentry>
        <flowentry>
          <id>ESCAPE-flowentry14</id>
          <port operation="replace">../../../ports/port[id=port-SAP14]</port>
          <out operation="replace">../../../NF_instances/node[id=comp]/ports/port[id=1]</out>
        </flowentry>
        <flowentry>
          <id>ESCAPE-flowentry15</id>
          <port operation="replace">../../../NF_instances/node[id=comp]/ports/port[id=1]</port>
          <out operation="replace">../../../NF_instances/node[id=decomp]/ports/port[id=1]</out>
        </flowentry>
```

```
        <flowentry  operation ="create">
          <id>ESCAPE-flowentry16</id>
          <name>sg_hop:16</name>
          <priority>100</priority>
          <port>../../../NF_instances/node[id=decomp]/ports/port[id=1]</port>
          <out>../../../ports/port[id=port-SAP1]</out>
        </flowentry>
      </flowtable>
    </node>
  </nodes>
  <version>2016-02-24; compiled at 2016-03-18 19:56:13</version>
</virtualizer>
```

The result global configurations configuration is showed in Listing 51, the Docker configuration in Listing 52, and the Mininet configuration in Listing 53.

Listing 51: NFFG of Third step result: global configuration: xml view

```
<?xml version="1.0" ?>
<virtualizer>
  <id>SingleBiSBiS-NFFG</id>
  <name>Single-BiSBiS-View</name>
  <nodes>
    <node>
      <id>SingleBiSBiS</id>
      <name>SingleBiSBiS</name>
      <type>BiSBiS</type>
      <ports>
        <port>
          <id>port-SAP1</id>
          <name>SAP1</name>
          <port_type>port-sap</port_type>
        </port>
        <port>
          <id>port-SAP2</id>
          <name>SAP2</name>
          <port_type>port-sap</port_type>
        </port>
        <port>
          <id>port-SAP55</id>
          <name>SAP55</name>
          <port_type>port-sap</port_type>
        </port>
      </ports>
      <links>
        <link>
          <id>resource-link0</id>
          <src>../../../ports/port[id=port-SAP55]</src>
          <dst>../../../ports/port[id=port-SAP2]</dst>
          <resources>
            <delay>0</delay>
            <bandwidth>5280000.0</bandwidth>
          </resources>
        </link>
        <link>
          <id>resource-link1</id>
          <src>../../../ports/port[id=port-SAP55]</src>
          <dst>../../../ports/port[id=port-SAP1]</dst>
          <resources>
            <delay>0</delay>
            <bandwidth>5280000.0</bandwidth>
```

```
            </resources>
          </link>
          <link>
            <id>resource-link2</id>
            <src>../../../ports/port[id=port-SAP2]</src>
            <dst>../../../ports/port[id=port-SAP1]</dst>
            <resources>
              <delay>0</delay>
              <bandwidth>5280000.0</bandwidth>
            </resources>
          </link>
        </links>
        <resources>
          <cpu>34.0</cpu>
          <mem>50721069066.0</mem>
          <storage>25010148362.0</storage>
        </resources>
        <metadata>
          <key>bandwidth</key>
          <value>5280000.0</value>
        </metadata>
        <metadata>
          <key>delay</key>
          <value>0</value>
        </metadata>
        <NF_instances>
          <node>
            <id>comp</id>
            <name>COMPRESSOR</name>
            <type>headerCompressor</type>
            <ports>
              <port>
                <id>1</id>
                <port_type>port-abstract</port_type>
              </port>
            </ports>
            <resources>
              <cpu>1.0</cpu>
              <mem>1.0</mem>
              <storage>0.0</storage>
            </resources>
          </node>
          <node>
            <id>decomp</id>
            <name>DECOMPRESSOR</name>
            <type>headerDecompressor</type>
            <ports>
              <port>
                <id>1</id>
                <port_type>port-abstract</port_type>
              </port>
            </ports>
            <resources>
              <cpu>1.0</cpu>
              <mem>1.0</mem>
              <storage>0.0</storage>
            </resources>
          </node>
          <node>
            <id>dockernf</id>
```

```
            <name>dockernf</name>
            <type> bridge </type>
            <ports>
              <port>
                <id>1</id>
                <port_type>port-abstract</port_type>
              </port>
              <port>
                <id>2</id>
                <port_type>port-abstract</port_type>
              </port>
            </ports>
            <resources>
              <cpu>1.0</cpu>
              <mem>1.0</mem>
              <storage>0.0</storage>
            </resources>
          </node>
          <node>
            <id>dpi</id>
            <name>DPI</name>
            <type>dpi</type>
            <ports>
              <port>
                <id>1</id>
                <port_type>port-abstract</port_type>
              </port>
              <port>
                <id>2</id>
                <port_type>port-abstract</port_type>
              </port>
            </ports>
            <resources>
              <cpu>1.0</cpu>
              <mem>1.0</mem>
              <storage>0.0</storage>
            </resources>
          </node>
          <node>
            <id>webserver1</id>
            <name>webserver1</name>
            <type>webserver</type>
            <ports>
              <port>
                <id>0</id>
                <port_type>port-abstract</port_type>
              </port>
            </ports>
            <resources>
              <cpu>1.0</cpu>
              <mem>1.0</mem>
              <storage>0.0</storage>
            </resources>
          </node>
        </NF_instances>
        < capabilities >
          <supported_NFs>
            <node>
              <id> bridge </id>
              <type> bridge </type>
```

```
            </node>
            <node>
              <id>dpi</id>
              <type>dpi</type>
            </node>
            <node>
              <id>headerCompressor</id>
              <type>headerCompressor</type>
            </node>
            <node>
              <id>headerDecompressor</id>
              <type>headerDecompressor</type>
            </node>
            <node>
              <id> simpleForwarder </id>
              <type> simpleForwarder </type>
            </node>
            <node>
              <id>webserver</id>
              <type>webserver</type>
            </node>
          </supported_NFs>
        </ capabilities >
        <flowtable>
          <flowentry>
            <id>ESCAPE—flowentry11</id>
            <name>sg_hop:11</name>
            < priority >100</ priority >
            <port >../../../ ports/port[id=port—SAP1]</port>
            <out >../../../ NF_instances/node[id=webserver1]/ports/port[id=0]</out>
            <resources>
              <bandwidth>1.0</bandwidth>
            </resources>
          </flowentry>
          <flowentry>
            <id>ESCAPE—flowentry12</id>
            <name>sg_hop:12</name>
            < priority >100</ priority >
            <port >../../../ NF_instances/node[id=webserver1]/ports/port[id=0]</port>
            <out >../../../ NF_instances/node[id=dpi]/ports/port[id=1]</out>
            <resources>
              <bandwidth>1.0</bandwidth>
            </resources>
          </flowentry>
          <flowentry>
            <id>ESCAPE—flowentry13</id>
            <name>sg_hop:13</name>
            < priority >100</ priority >
            <port >../../../ NF_instances/node[id=dpi]/ports/port[id=2]</port>
            <out >../../../ NF_instances/node[id=dockernf]/ports/port[id=1]</out>
            <resources>
              <bandwidth>1.0</bandwidth>
            </resources>
          </flowentry>
          <flowentry>
            <id>ESCAPE—flowentry14</id>
            <name>sg_hop:14</name>
            < priority >100</ priority >
            <port >../../../ NF_instances/node[id=dockernf]/ports/port[id=2]</port>
            <out >../../../ NF_instances/node[id=comp]/ports/port[id=1]</out>
```

```
            <resources>
              <bandwidth>1.0</bandwidth>
            </resources>
          </flowentry>
          <flowentry>
            <id>ESCAPE-flowentry15</id>
            <name>sg_hop:15</name>
            < priority >100</ priority >
            <port >../../../ NF_instances/node[id=comp]/ports/port[id =1]</ port>
            <out >../../../ NF_instances/node[id=decomp]/ports/port[id =1]</ out>
            <resources>
              <bandwidth>1.0</bandwidth>
            </resources>
          </flowentry>
          <flowentry>
            <id>ESCAPE-flowentry16</id>
            <name>sg_hop:16</name>
            < priority >100</ priority >
            <port >../../../ NF_instances/node[id=decomp]/ports/port[id =1]</ port>
            <out >../../../ ports/port[id=port-SAP1]</out>
            <resources>
              <bandwidth>1.0</bandwidth>
            </resources>
          </flowentry>
        </flowtable>
      </node>
    </nodes>
    < version >2016-02-24; compiled at 2016-03-18 19:56:13</ version >
</ virtualizer >
```

Listing 52: NFFG of Third step result: Docker configuration: xml view

```
<?xml version = " 1.0 " ?>
< virtualizer >
  <id>DOCKER</id>
  <name>Docker domain</name>
  <nodes>
    <node>
      <id>DOCKER_HOST</id>
      <name>atitude-Linux</name>
      <type> BiSBiS </type>
      <ports>
        <port>
          <id>eth0</id>
          <name>SAP34</name>
          <port_type>port-sap</port_type>
          <sap>SAP34</sap>
        </port>
      </ports>
      <resources>
        <cpu>4</cpu>
        <mem>50721017856</mem>
        <storage>25010147328</storage>
      </resources>
      <NF_instances>
        <node>
          <id> dockernf </id>
          <name>dockernf</name>
          <type> bridge </type>
          <ports>
```

```
            <port>
              <id>1</id>
              <port_type>port-abstract</port_type>
            </port>
            <port>
              <id>2</id>
              <port_type>port-abstract</port_type>
            </port>
          </ports>
          <resources>
            <cpu>1.0</cpu>
            <mem>1.0</mem>
            <storage>0.0</storage>
          </resources>
        </node>
      </NF_instances>
      < capabilities >
        <supported_NFs>
          <node>
            <id>dockernf</id>
            <name>dockernf</name>
            <type>bridge</type>
            <ports>
              <port>
                <id>eth0</id>
                <name>mgmt</name>
                <port_type> public -ports</port_type>
                <metadata>
                  <key> localports </key>
                  <value>22</value>
                </metadata>
              </port>
              <port>
                <id>eth1</id>
                <name>input</name>
                <port_type>port-abstract</port_type>
              </port>
              <port>
                <id>eth2</id>
                <name>output</name>
                <port_type>port-abstract</port_type>
              </port>
            </ports>
            <resources>
              <cpu>1</cpu>
              <mem>128</mem>
              <storage>1</storage>
            </resources>
          </node>
        </supported_NFs>
      </ capabilities >
      <flowtable>
        <flowentry>
          <id>ESCAPE-flowentry13</id>
          <name>sg_hop:13</name>
          < priority >100</ priority >
          <port >../../../ ports/port[id=eth0]</port>
          <match>dl_tag=0x000d</match>
          <action>pop_tag</action>
          <out >../../../ NF_instances/node[id=dockernf]/ports/port[id=1]</out>
```

```
            <resources>
              <bandwidth>1.0</bandwidth>
            </resources>
          </flowentry>
          <flowentry>
            <id>ESCAPE-flowentry14</id>
            <name>sg_hop:14</name>
            < priority >100</ priority >
            <port >../../../ NF_instances/node[id=dockernf]/ports/port[id=2]</port>
            <action>push_tag:0x000e</action>
            <out >../../../ ports/port[id=eth0]</out>
            <resources>
              <bandwidth>1.0</bandwidth>
            </resources>
          </flowentry>
        </flowtable>
      </node>
    </nodes>
    <metadata>
      <key>REQ:5</key>
      <value >{'snode':'SAP34','sg_path':[13,14],'delay':'0.000','bw':'0.000','dport':'eth0','sport':'eth0','dnode':'SAP34'}</value>
    </metadata>
    < version >2016-02-24; compiled at 2016-03-18 19:56:13</ version >
</ virtualizer >
```

Listing 53: NFFG of Third step result: Mininet configuration: xml view

```
<?xml version="1.0" ?>
< virtualizer >
  <id> SingleBiSBiS -NFFG</id>
  <name>Single-BiSBiS-View</name>
  <nodes>
    <node>
      <id> SingleBiSBiS </id>
      <name>SingleBiSBiS</name>
      <type>BiSBiS</type>
      <ports>
        <port>
          <id>port-SAP1</id>
          <name>SAP1</name>
          <port_type>port-sap</port_type>
        </port>
        <port>
          <id>port-SAP14</id>
          <name>SAP14</name>
          <port_type>port-sap</port_type>
          <sap>SAP14</sap>
        </port>
        <port>
          <id>port-SAP2</id>
          <name>SAP2</name>
          <port_type>port-sap</port_type>
        </port>
      </ports>
      < links >
        <link>
          <id>resource-link0</id>
          <src >../../../ ports/port[id=port-SAP2]</src>
          <dst >../../../ ports/port[id=port-SAP14]</dst>
          <resources>
```

```
            <delay>0.2</delay>
            <bandwidth>200000.0</bandwidth>
          </resources>
        </link>
        <link>
          <id>resource-link1</id>
          <src>../../../ports/port[id=port-SAP2]</src>
          <dst>../../../ports/port[id=port-SAP1]</dst>
          <resources>
            <delay>0.2</delay>
            <bandwidth>200000.0</bandwidth>
          </resources>
        </link>
        <link>
          <id>resource-link2</id>
          <src>../../../ports/port[id=port-SAP14]</src>
          <dst>../../../ports/port[id=port-SAP1]</dst>
          <resources>
            <delay>0.2</delay>
            <bandwidth>200000.0</bandwidth>
          </resources>
        </link>
      </links>
      <resources>
        <cpu>10.0</cpu>
        <mem>10.0</mem>
        <storage>10.0</storage>
      </resources>
      <metadata>
        <key>bandwidth</key>
        <value>200000.0</value>
      </metadata>
      <metadata>
        <key>delay</key>
        <value>0.2</value>
      </metadata>
      <NF_instances>
        <node>
          <id>comp</id>
          <name>COMPRESSOR</name>
          <type>headerCompressor</type>
          <ports>
            <port>
              <id>1</id>
              <port_type>port-abstract</port_type>
            </port>
          </ports>
          <resources>
            <cpu>1.0</cpu>
            <mem>1.0</mem>
            <storage>0.0</storage>
          </resources>
        </node>
        <node>
          <id>decomp</id>
          <name>DECOMPRESSOR</name>
          <type>headerDecompressor</type>
          <ports>
            <port>
              <id>1</id>
```

```
94                <port_type>port—abstract</port_type>
95              </port>
96            </ports>
97            <resources>
98              <cpu>1.0</cpu>
99              <mem>1.0</mem>
100             <storage>0.0</storage>
101           </resources>
102         </node>
103       </NF_instances>
104       < capabilities >
105         <supported_NFs>
106           <node>
107             <id>headerCompressor</id>
108             <type>headerCompressor</type>
109           </node>
110           <node>
111             <id>headerDecompressor</id>
112             <type>headerDecompressor</type>
113           </node>
114           <node>
115             <id> simpleForwarder </id>
116             <type> simpleForwarder </type>
117           </node>
118         </supported_NFs>
119       </ capabilities >
120       <flowtable>
121         <flowentry>
122           <id>ESCAPE—flowentry11</id>
123           <name>sg_hop:11</name>
124           < priority >100</ priority >
125           <port >../../../ ports/port[id=port—SAP1]</port>
126           <out >../../../ ports/port[id=port—SAP14]</out>
127           <resources>
128             <bandwidth>0.0</bandwidth>
129           </resources>
130         </flowentry>
131         <flowentry>
132           <id>ESCAPE—flowentry14</id>
133           <name>sg_hop:14</name>
134           < priority >100</ priority >
135           <port >../../../ ports/port[id=port—SAP14]</port>
136           <out >../../../ NF_instances/node[id=comp]/ports/port[id =1]</ out>
137           <resources>
138             <bandwidth>1.0</bandwidth>
139           </resources>
140         </flowentry>
141         <flowentry>
142           <id>ESCAPE—flowentry15</id>
143           <name>sg_hop:15</name>
144           < priority >100</ priority >
145           <port >../../../ NF_instances/node[id=comp]/ports/port[id =1]</ port>
146           <out >../../../ NF_instances/node[id=decomp]/ports/port[id =1]</ out>
147           <resources>
148             <bandwidth>1.0</bandwidth>
149           </resources>
150         </flowentry>
151         <flowentry>
152           <id>ESCAPE—flowentry16</id>
153           <name>sg_hop:16</name>
```

```
154             < priority >100</ priority >
155             <port >../../../ NF_instances/node[id=decomp]/ports/port[id =1]</ port>
156             <out >../../../ ports/port[id=port-SAP1]</out>
157             <resources>
158               <bandwidth>1.0</bandwidth>
159             </resources>
160           </flowentry>
161         </flowtable>
162       </node>
163     </nodes>
164     < version >2016-02-24; compiled at 2016-03-18 19:56:13</ version >
165 </ virtualizer >
```

A.7 Service Request 5: Additional port to bridge NF

In the fifth example we extend the existing Ethernet bridge NF with an additional port, and forward the output of this port to User3. This will result mirroring all traffic going to User1 to User3 as well. The fifth service request is depicted in Fig. 43, and formalized in Listing 54 to be sent to ESCAPE.

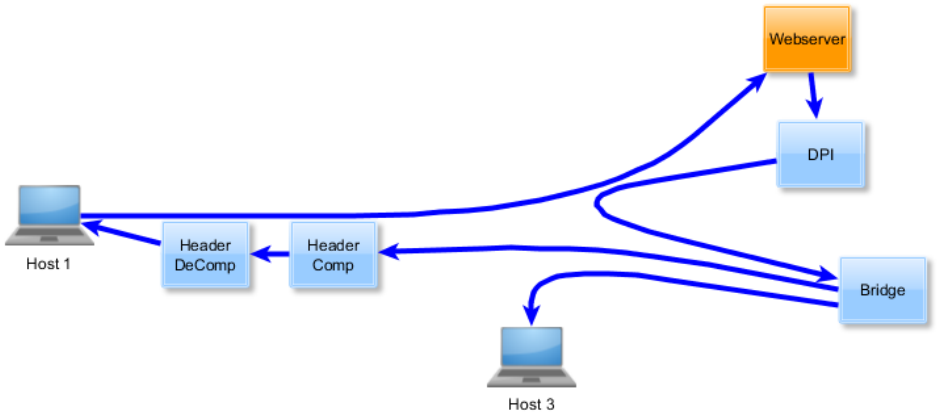


Figure 43: request5

Listing 54: Fifth service request (service graph) to ESCAPE

```
{
  "parameters": {
    "id": "EWSDN-demo-req3",
    "name": "EWSDN-2web-1dpi-1comp-1decomp-2SAP-req",
    "version": "1.0"
  },
  "node_nfs": [
    {
      "id": "webserver1",
      "name": "webserver1",
      "ports": [
        {
          "id": 0
        }
      ],
      "functional_type": "webserver",
      "specification": {
        "resources": {
          "cpu": 1,
          "mem": 1,
          "storage": 0
        }
      }
    },
    {
      "id": "dpi",
      "name": "DPI",
      "ports": [
        {
          "id": 1
        },
        {
          "id": 2
```

```
        }
      ],
      "functional_type": "dpi",
      "specification": {
        "resources": {
          "cpu": 1,
          "mem": 1,
          "storage": 0
        }
      }
    },
    {
      "id": "comp",
      "name": "COMPRESSOR",
      "ports": [
        {
          "id": 1
        }
      ],
      "functional_type": "headerCompressor",
      "specification": {
        "resources": {
          "cpu": 1,
          "mem": 1,
          "storage": 0
        }
      }
    },
    {
      "id": "decomp",
      "name": "DECOMPRESSOR",
      "ports": [
        {
          "id": 1
        }
      ],
      "functional_type": "headerDecompressor",
      "specification": {
        "resources": {
          "cpu": 1,
          "mem": 1,
          "storage": 0
        }
      }
    },
    {
      "id": "dockernf",
      "name": "dockernf",
      "ports": [
        {
          "id": 1
        },
        {
          "id": 2
        },
        {
          "id": 3
        }
      ],
      "functional_type": "bridge",
```

```
        "  specification  ": {
          " resources ": {
            "cpu": 1,
            "mem": 1,
            " storage ": 0
          }
        }
      }
    ],
    "node_saps ": [
      {
        " id ": "sap2",
        "name": "SAP2",
        " ports ": [
          {
            " id ": 1
          }
        ]
      },
      {
        " id ": "sap1",
        "name": "SAP1",
        " ports ": [
          {
            " id ": 1
          }
        ]
      },
      {
        " id ": "sap55",
        "name": "SAP55",
        " ports ": [
          {
            " id ": 1
          }
        ]
      }
    ],
    " edge_sg_nexthops ": [
      {
        " id ": 11,
        " src_node ": "sap1",
        " src_port ": 1,
        " dst_node ": "webserver1",
        " dst_port ": 0
      },
      {
        " id ": 12,
        " src_node ": "webserver1",
        " src_port ": 0,
        " dst_node ": "dpi",
        " dst_port ": 1
      },
      {
        " id ": 13,
        " src_node ": "dpi",
        " src_port ": 2,
        " dst_node ": "dockernf",
        " dst_port ": 1
      },
```

```
    {
      "id": 14,
      "src_node": "dockernf",
      "src_port": 2,
      "dst_node": "comp",
      "dst_port": 1
    },

    {
      "id": 15,
      "src_node": "comp",
      "src_port": 1,
      "dst_node": "decomp",
      "dst_port": 1
    },
    {
      "id": 16,
      "src_node": "decomp",
      "src_port": 1,
      "dst_node": "sap1",
      "dst_port": 1
    },
   {
      "id": 17,
      "src_node": "dockernf",
      "src_port": 3,
      "dst_node": "sap55",
      "dst_port": 1
    }
  ],
  "edge_reqs": [
    {
      "id": 36246224,
      "src_node": "sap1",
      "src_port": 1,
      "dst_node": "sap1",
      "dst_port": 1,
      "delay": 100,
      "bandwidth": 1,
      "sg_path": [
        11,
        12,
        13,
        14,
        15,
        16
      ]
    }
  ]
}
```

The above service request can be formalized with the Virtualizer model as described in Listing 55.

Listing 55: NFFG of Fifth request (Virtualizer) to ESCAPE: xml view

```
<?xml version="1.0" ?>
<virtualizer>
  <nodes>
    <node>
      <id>SingleBiSBiS</id>
      <NF_instances>
```

```
        <node>
          <id>dockernf</id>
          <ports>
            <port operation="create">
              <id>3</id>
              <port_type>port-abstract</port_type>
            </port>
          </ports>
        </node>
      </NF_instances>
      <flowtable>
        <flowentry operation="create">
          <id>ESCAPE-flowentry17</id>
          <name>sg_hop:17</name>
          <priority>100</priority>
          <port>../../../NF_instances/node[id=dockernf]/ports/port[id=3]</port>
          <out>../../../ports/port[id=port-SAP55]</out>
        </flowentry>
      </flowtable>
    </node>
  </nodes>
  <version>2016-02-24; compiled at 2016-03-18 19:56:13</version>
</virtualizer>
```

The outcome of the global orchestration is shown in Fig. 44.

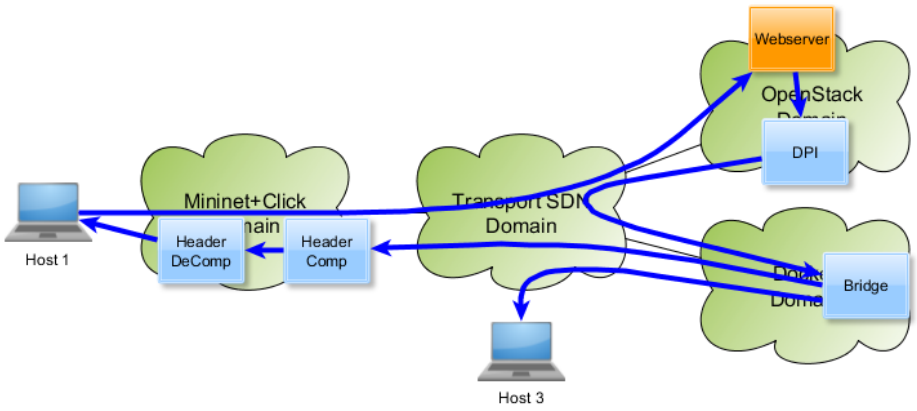


Figure 44: deployed5

The configuration messages sent by the global orchestrator to the Docker domain is visible in Listing 56.

Listing 56: NFFG of Third step configuration message to Docker domain: xml view

```
<?xml version="1.0" ?>
<virtualizer>
  <nodes>
    <node>
      <id>DOCKER_HOST</id>
      <NF_instances>
        <node>
          <id>dockernf</id>
          <ports>
            <port operation="create">
              <id>3</id>
              <port_type>port-abstract</port_type>
            </port>
```

```
            </ports>
          </node>
        </NF_instances>
        <flowtable>
          <flowentry operation="create">
            <id>ESCAPE-flowentry17</id>
            <name>sg_hop:17</name>
            <priority>100</priority>
            <port>../../../NF_instances/node[id=dockernf]/ports/port[id=3]</port>
            <action>push_tag:0x0011</action>
            <out>../../../ports/port[id=eth0]</out>
          </flowentry>
        </flowtable>
      </node>
    </nodes>
    <version>2016-02-24; compiled at 2016-03-18 19:56:13</version>
</virtualizer>
```

The result global configurations configuration is showed in Listing 57, and the Docker configuration in Listing 58.

Listing 57: NFFG of Third step result: global configuration: xml view

```
<?xml version="1.0"?>
<virtualizer>
  <id>SingleBiSBiS-NFFG</id>
  <name>Single-BiSBiS-View</name>
  <nodes>
    <node>
      <id>SingleBiSBiS</id>
      <name>SingleBiSBiS</name>
      <type>BiSBiS</type>
      <ports>
        <port>
          <id>port-SAP1</id>
          <name>SAP1</name>
          <port_type>port-sap</port_type>
        </port>
        <port>
          <id>port-SAP2</id>
          <name>SAP2</name>
          <port_type>port-sap</port_type>
        </port>
        <port>
          <id>port-SAP55</id>
          <name>SAP55</name>
          <port_type>port-sap</port_type>
        </port>
      </ports>
      <links>
        <link>
          <id>resource-link0</id>
          <src>../../../ports/port[id=port-SAP55]</src>
          <dst>../../../ports/port[id=port-SAP2]</dst>
          <resources>
            <delay>0</delay>
            <bandwidth>39200000.0</bandwidth>
          </resources>
        </link>
        <link>
          <id>resource-link1</id>
          <src>../../../ports/port[id=port-SAP55]</src>
```

```
            <dst  >../../../  ports/port[id=port-SAP1]</dst>
            <resources>
              <delay>0</delay>
              <bandwidth>39200000.0</bandwidth>
            </resources>
          </link>
          <link>
            <id>resource-link2</id>
            <src  >../../../  ports/port[id=port-SAP2]</src>
            <dst  >../../../  ports/port[id=port-SAP1]</dst>
            <resources>
              <delay>0</delay>
              <bandwidth>39200000.0</bandwidth>
            </resources>
          </link>
        </links>
        <resources>
          <cpu>34.0</cpu>
          <mem>50721069066.0</mem>
          <storage>25010148362.0</storage>
        </resources>
        <metadata>
          <key>bandwidth</key>
          <value>39200000.0</value>
        </metadata>
        <metadata>
          <key>delay</key>
          <value>0</value>
        </metadata>
        <NF_instances>
          <node>
            <id>comp</id>
            <name>COMPRESSOR</name>
            <type>headerCompressor</type>
            <ports>
              <port>
                <id>1</id>
                <port_type>port-abstract</port_type>
              </port>
            </ports>
            <resources>
              <cpu>1.0</cpu>
              <mem>1.0</mem>
              <storage>0.0</storage>
            </resources>
          </node>
          <node>
            <id>decomp</id>
            <name>DECOMPRESSOR</name>
            <type>headerDecompressor</type>
            <ports>
              <port>
                <id>1</id>
                <port_type>port-abstract</port_type>
              </port>
            </ports>
            <resources>
              <cpu>1.0</cpu>
              <mem>1.0</mem>
              <storage>0.0</storage>
```

```
      </resources>
    </node>
    <node>
      <id>dockernf</id>
      <name>dockernf</name>
      <type>bridge</type>
      <ports>
        <port>
          <id>1</id>
          <port_type>port-abstract</port_type>
        </port>
        <port>
          <id>2</id>
          <port_type>port-abstract</port_type>
        </port>
        <port>
          <id>3</id>
          <port_type>port-abstract</port_type>
        </port>
      </ports>
      <resources>
        <cpu>1.0</cpu>
        <mem>1.0</mem>
        <storage>0.0</storage>
      </resources>
    </node>
    <node>
      <id>dpi</id>
      <name>DPI</name>
      <type>dpi</type>
      <ports>
        <port>
          <id>1</id>
          <port_type>port-abstract</port_type>
        </port>
        <port>
          <id>2</id>
          <port_type>port-abstract</port_type>
        </port>
      </ports>
      <resources>
        <cpu>1.0</cpu>
        <mem>1.0</mem>
        <storage>0.0</storage>
      </resources>
    </node>
    <node>
      <id>webserver1</id>
      <name>webserver1</name>
      <type>webserver</type>
      <ports>
        <port>
          <id>0</id>
          <port_type>port-abstract</port_type>
        </port>
      </ports>
      <resources>
        <cpu>1.0</cpu>
        <mem>1.0</mem>
        <storage>0.0</storage>
```

```
        </resources>
      </node>
    </NF_instances>
    < capabilities >
      <supported_NFs>
        <node>
          <id> bridge </id>
          <type> bridge </type>
        </node>
        <node>
          <id> dpi </id>
          <type> dpi </type>
        </node>
        <node>
          <id>headerCompressor</id>
          <type>headerCompressor</type>
        </node>
        <node>
          <id>headerDecompressor</id>
          <type>headerDecompressor</type>
        </node>
        <node>
          <id> simpleForwarder </id>
          <type> simpleForwarder </type>
        </node>
        <node>
          <id>webserver</id>
          <type>webserver</type>
        </node>
      </supported_NFs>
    </ capabilities >
    <flowtable>
      <flowentry>
        <id>ESCAPE—flowentry11</id>
        <name>sg_hop:11</name>
        < priority >100</ priority >
        <port >../../../ ports/port[id=port—SAP1]</port>
        <out >../../../ NF_instances/node[id=webserver1]/ports/port[id=0]</out>
        <resources>
          <bandwidth>1.0</bandwidth>
        </resources>
      </flowentry>
      <flowentry>
        <id>ESCAPE—flowentry12</id>
        <name>sg_hop:12</name>
        < priority >100</ priority >
        <port >../../../ NF_instances/node[id=webserver1]/ports/port[id=0]</port>
        <out >../../../ NF_instances/node[id=dpi]/ports/port[id=1]</out>
        <resources>
          <bandwidth>1.0</bandwidth>
        </resources>
      </flowentry>
      <flowentry>
        <id>ESCAPE—flowentry13</id>
        <name>sg_hop:13</name>
        < priority >100</ priority >
        <port >../../../ NF_instances/node[id=dpi]/ports/port[id=2]</port>
        <out >../../../ NF_instances/node[id=dockernf]/ports/port[id=1]</out>
        <resources>
          <bandwidth>1.0</bandwidth>
```

```
            </resources>
          </flowentry>
          <flowentry>
            <id>ESCAPE—flowentry14</id>
            <name>sg_hop:14</name>
            < priority >100</ priority >
            <port >../../../ NF_instances/node[id=dockernf]/ports/port[id=2]</port>
            <out >../../../ NF_instances/node[id=comp]/ports/port[id=1]</out>
            <resources>
              <bandwidth>1.0</bandwidth>
            </resources>
          </flowentry>
          <flowentry>
            <id>ESCAPE—flowentry15</id>
            <name>sg_hop:15</name>
            < priority >100</ priority >
            <port >../../../ NF_instances/node[id=comp]/ports/port[id=1]</port>
            <out >../../../ NF_instances/node[id=decomp]/ports/port[id=1]</out>
            <resources>
              <bandwidth>1.0</bandwidth>
            </resources>
          </flowentry>
          <flowentry>
            <id>ESCAPE—flowentry16</id>
            <name>sg_hop:16</name>
            < priority >100</ priority >
            <port >../../../ NF_instances/node[id=decomp]/ports/port[id=1]</port>
            <out >../../../ ports/port[id=port—SAP1]</out>
            <resources>
              <bandwidth>1.0</bandwidth>
            </resources>
          </flowentry>
          <flowentry>
            <id>ESCAPE—flowentry17</id>
            <name>sg_hop:17</name>
            < priority >100</ priority >
            <port >../../../ NF_instances/node[id=dockernf]/ports/port[id=3]</port>
            <out >../../../ ports/port[id=port—SAP55]</out>
            <resources>
              <bandwidth>0</bandwidth>
            </resources>
          </flowentry>
        </flowtable>
      </node>
    </nodes>
    < version >2016—02—24; compiled at 2016—03—18 19:56:13</ version >
</ virtualizer >
```

Listing 58: NFFG of Third step result: Docker configuration: xml view

```
<?xml version = " 1.0 " ?>
< virtualizer >
  <id>DOCKER</id>
  <name>Docker domain</name>
  <nodes>
    <node>
      <id>DOCKER_HOST</id>
      <name>atitude—Linux</name>
      <type> BiSBiS </type>
      <ports>
```

```
      <port>
        <id>eth0</id>
        <name>SAP34</name>
        <port_type>port-sap</port_type>
        <sap>SAP34</sap>
      </port>
    </ports>
    <resources>
      <cpu>4</cpu>
      <mem>50721017856</mem>
      <storage>25010147328</storage>
    </resources>
    <NF_instances>
      <node>
        <id> dockernf </id>
        <name>dockernf</name>
        <type> bridge </type>
        <ports>
          <port>
            <id>1</id>
            <port_type>port-abstract</port_type>
          </port>
          <port>
            <id>2</id>
            <port_type>port-abstract</port_type>
          </port>
          <port>
            <id>3</id>
            <port_type>port-abstract</port_type>
          </port>
        </ports>
        <resources>
          <cpu>1.0</cpu>
          <mem>1.0</mem>
          <storage>0.0</storage>
        </resources>
      </node>
    </NF_instances>
    < capabilities >
      <supported_NFs>
        <node>
          <id> dockernf </id>
          <name>dockernf</name>
          <type> bridge </type>
          <ports>
            <port>
              <id>eth0</id>
              <name>mgmt</name>
              <port_type> public -ports</port_type>
              <metadata>
                <key> localports </key>
                <value>22</value>
              </metadata>
            </port>
            <port>
              <id>eth1</id>
              <name>input</name>
              <port_type>port-abstract</port_type>
            </port>
            <port>
```

```
                    <id>eth2</id>
                    <name>output</name>
                    <port_type>port-abstract</port_type>
                  </port>
                </ports>
                <resources>
                  <cpu>1</cpu>
                  <mem>128</mem>
                  <storage>1</storage>
                </resources>
              </node>
            </supported_NFs>
          </ capabilities >
          <flowtable>
            <flowentry>
              <id>ESCAPE-flowentry13</id>
              <name>sg_hop:13</name>
              < priority >100</ priority >
              <port >../../../ ports/port[id=eth0]</port>
              <match>dl_tag=0x000d</match>
              <action>pop_tag</action>
              <out >../../../ NF_instances/node[id=dockernf]/ports/port[id=1]</out>
              <resources>
                <bandwidth>1.0</bandwidth>
              </resources>
            </flowentry>
            <flowentry>
              <id>ESCAPE-flowentry14</id>
              <name>sg_hop:14</name>
              < priority >100</ priority >
              <port >../../../ NF_instances/node[id=dockernf]/ports/port[id=2]</port>
              <action>push_tag:0x000e</action>
              <out >../../../ ports/port[id=eth0]</out>
              <resources>
                <bandwidth>1.0</bandwidth>
              </resources>
            </flowentry>
            <flowentry>
              <id>ESCAPE-flowentry17</id>
              <name>sg_hop:17</name>
              < priority >100</ priority >
              <port >../../../ NF_instances/node[id=dockernf]/ports/port[id=3]</port>
              <action>push_tag:0x0011</action>
              <out >../../../ ports/port[id=eth0]</out>
              <resources>
                <bandwidth>0</bandwidth>
              </resources>
            </flowentry>
          </flowtable>
        </node>
      </nodes>
      <metadata>
        <key>REQ:5</key>
        <value >{' snode ':' SAP34 ',' sg_path ':[13,14],' delay ':'0.000',' bw ':'0.000',' dport ':' eth0 ',' sport ':' eth0 ',' dnode ':' SAP34 '}</value>
      </metadata>
      < version >2016-02-24; compiled at 2016-03-18 19:56:13</ version >
    </ virtualizer >
```

References

[Blo08] Vincent D Blondel et. al. "Fast unfolding of communities in large networks". In: Journal of Statistical Mechanics: Theory and Experiment 2008.10 (2008), P10008.

[Cadvisor] Google. cAdvisor. 2014. url: https://github.com/google/cadvisor.

[CRB] Mosharaf Chowdhury, Muntasir Raihan Rahman, and Raouf Boutaba. "ViNEYard: Virtual Network Embedding Algorithms with Coordinated Node and Link Mapping". In: IEEE/ACM Trans. Netw. ().

[CRB09] N. M. M. K. Chowdhury, M. R. Rahman, and R. Boutaba. "Virtual network embedding with coordinated node and link mapping". In: IEEE INFOCOM. Apr. 2009.

[D2.1] Mario Kind et al. Deliverable 2.1: Use Cases and Initial Architecture. Tech. rep. UNIFY Project, 2014.

[D2.2] Robert Szabo, Balazs Sonkoly, Mario Kind et al. Deliverable 2.2: Final Architecture. Tech. rep. UNIFY Project, 2014.

[D3.1] Wouter Tavernier et al. D3.1 Programmability framework. Tech. rep. D3.1. UNIFY Project, Oct. 2014.

[D3.2] Pontus Skoldstrom et al. D3.2 Detailed functional specification and algorithm description. Tech. rep. D3.2. UNIFY Project, Apr. 2015.

[D3.2a] Pontus Skoldstrom et al. D3.2a Network Function Forwarding Graph specification (Supplement to D3.2). Tech. rep. D3.2sup. UNIFY Project, July 2015.

[D3.3] Jokin Garay et al. D3.3 Revised framework with functions and semantics. Tech. rep. D3.3. UNIFY Project, Oct. 2015.

[D3.4] D3.4 Prototype deliverable. Tech. rep. D3.4. UNIFY Project, Oct. 2015.

[D4.2] Rebecca Steinert, Wolfgang John et al. Deliverable 4.2: Proposal for SP-DevOps network capabilities and tools. Tech. rep. D4.2. UNIFY Project, Sept. 2015.

[D4.3] Deliverable 4.3: Updated concept and evaluation results for SP-DevOps. Tech. rep. D4.3. UNIFY Project, 2016.

[D4.4] Deliverable 4.4: DevOpsPro code base. Tech. rep. D4.4. UNIFY Project, 2016.

[D5.5] D5.5 Universal Node Prototype. Tech. rep. UNIFY Project, 2015.

[Docker] Docker. Docker. 2016. url: https://www.docker.com/.

[ETS14a] ETSI GS NFV. "Network Functions Virtualisation (NFV); Architectural Framework". Dec. 2014. url: http://www.etsi.org/deliver/etsi_gs/NFV/001_099/002/01.02.01_60/gs_NFV002v010201p.pdf.

[ETS14b] ETSI GS NFV. "Virtual Network Functions Architecture". Dec. 2014. url: http://www.etsi.org/deliver/etsi%5C_gs/NFV-SWA/001%5C_099/001/01.01.01%5C_60/gs%5C_NFV-SWA001v010101p.pdf.

[ETS14c] ETSI GS NFV-MAN. "Network Functions Virtualisation (NFV); Management and Orchestration ". Dec. 2014. url: http://www.etsi.org/deliver/etsi_gs/NFV-MAN/001_099/001/01.01.01_60/gs_NFV-MAN001v010101p.pdf.

[ETS16] ETSI GS NFV-TST. "Network Functions Virtualisation (NFV); Pre-deployment Testing; Report on Validation of NFV Environments and Services ". Apr. 2016. url: http://www.etsi.org/deliver/etsi_gs/NFV-TST/001_099/001/01.01.01_60/gs_NFV-TST001v010101p.pdf.

[Fis+13] A. Fischer, M. Beck J. Botero, H. De Meer, and X. Hesselbach. "Virtual network embedding: A survey". In: IEEE Communications Surveys Tutorials 15.4 (2013).

[FSF] Carlo Fuerst, Stefan Schmid, and Anja Feldmann. "Virtual network embedding with collocation: Benefits and limitations of pre-clustering". In: IEEE 2nd International Conference on Cloud Networking, CloudNet 2013, San Francisco, CA, USA, November 11-13, 2013.

[Gem+14] Aaron Gember-Jacobson, Raajay Viswanathan, Chaithan Prakash, Robert Grandl, Junaid Khalid, Sourav Das, and Aditya Akella. "OpenNF: Enabling Innovation in Network Function Control". In: Proceedings of the 2014 ACM Conference on SIGCOMM. SIGCOMM '14. Chicago, Illinois, USA: ACM, 2014, pp. 163–174. isbn: 978-1-4503-2836-4. doi: 10.1145/2619239.2626313. url: http://doi.acm.org/10.1145/2619239.2626313.

[Han+15] Bo Han, V. Gopalakrishnan, Lusheng Ji, and Seungjoon Lee. "Network function virtualization: Challenges and opportunities for innovations". In: Communications Magazine, IEEE 53.2 (Feb. 2015), pp. 90–97. issn: 0163-6804. doi: 10.1109/MCOM.2015.7045396.

[Joh+15] Wolfgang John, Catalin Meirosu, Bertrand Pechenot, Pontus Skoldstrom, Per Kreuger, and Rebecca Steinert. "Scalable software defined monitoring for service provider devops". In: Software Defined Networks (EWSDN), 2015 Fourth European Workshop on. IEEE. 2015, pp. 61–66.

[KDS15] Babu Kothandaraman, Manxing Du, and Pontus Sköldström. "Centrally Controlled Distributed VNF State Management". In: Proceedings of the 2015 ACM SIGCOMM Workshop on Hot Topics in Middleboxes and Network Function Virtualization. ACM. 2015, pp. 37–42.

[M4.3] Pontus Sköldström et al. M4.3 Update on SP DevOps with focus on automated tools. Tech. rep. M4.3. UNIFY Project, Oct. 2015.

[Mat+14] Jon Matias, Jokin Garay, Alaitz Mendiola, Nerea Toledo, and Eduardo Jacob. "FlowNAC: Flow-based network access control". In: Software Defined Networks (EWSDN), 2014 Third European Workshop on. IEEE. 2014, pp. 79–84.

[Mat+15] J. Matias, J. Garay, N. Toledo, J. Unzilla, and E. Jacob. "Toward an SDN-enabled NFV architecture". In: Communications Magazine, IEEE 53.4 (Apr. 2015), pp. 187–193. issn: 0163-6804. doi: 10.1109/MCOM.2015.7081093.

[Mor16] A. Morton. Considerations for Benchmarking Virtual Network Functions and Their Infrastructure. 2016. url: https://datatracker.ietf.org/doc/draft-morton-bmwg-virtual-net/.

[OpenTSDB] The OpenTSDB Authors. OpenTSDB. 2010. url: http://opentsdb.net/.

[OPN16] OPNFV. Project: Yardstick - Infrastructure Verification. 2016. url: https://wiki.opnfv.org/display/yardstick/.

[Orl+07] S. Orlowski, M. Pióro, A. Tomaszewski, and R. Wessäly. "SNDlib 1.0–Survivable Network Design Library". In: Proceedings of the 3rd International Network Optimization Conference (INOC 2007), Spa, Belgium. 2007.

[Pai+15] E. Paik, M-K. Shin, S. Pack, and S. Lee. Resource Management in Service Chaining. 2015. url: http://tools.ietf.org/html/draft-lee-nfvrg-resource-management-service-chain-01.

[PipelineDB] PipelineDB. PipelineDB. 2016. url: https://www.pipelinedb.com/.

[SQK16] R. Szabo, Z. Qiang, and M. Kind. Recursive virtualization and programming for network and cloud resources. 2016. url: https://datatracker.ietf.org/doc/draft-irtf-nfvrg-unify-recursive-programming/.

[TOM16] M. Tahhan, B. O'Mahony, and A. Morton. Benchmarking Virtual Switches in OPNFV. 2016. url: https://datatracker.ietf.org/doc/draft-vsperf-bmwg-vswitch-opnfv/.