PADERBORN UNIVERSITY
The University for the Information Society

Department of Computer Science
Computer Networks Research Group

Project Report

SCrAMbLE

Management of ServiCes Across MultipLE clouds

Authors:

Arkajit Dhar
Ashwin Prasad Shivarpatna Venkatesh
Bhargavi Mohan
Deeksha Mysore Ramesh
Sanket Kumar Gupta
Suheel Shrirangapura Nazeersab
Vivek Jaganath

Supervisors:

Prof. Dr. Holger Karl | Sevil Dräxler | Hadi Razzaghi Kouchaksaraei

Paderborn, September 19, 2019

Contents

TODOs

List of Figures

List of Tables

1 Motivation

The rapid growth of mobile data services driven by mobile internet has led to substantial challenges of high availability, low latency, high bit rate and performances in networks. The recent development of Network Function Virtualization (NFV) and Software-Defined Networks (SDN) have emerged as key enablers for 5G networks. There has been a paradigm shift in networking with the recent developments in network virtualization technology. NFV involves decoupling of the hardware components from the software components of a network function. NFV requires a central management and orchestrations (MANO) framework in order to fully deliver end-to-end services of an application. There are multiple open source as well as industrial implementations of MANO framework in the market.

End-to-end network service delivery requires chaining of the Virtual Network Functions (VNFs) across different Internet Service Providers (ISPs) which in turn have their own MANO frameworks. In order to seamlessly create a network service by utilizing the VNF within each of the MANO frameworks, the need of interoperability among different MANO frameworks is of utmost importance.

1.1 Problem Description

European Telecommunications Standards Institute (ETSI) defines the reference architecture for a MANO framework. Each network service is composed of multiple network functions virtually linked and orchestrated by a MANO framework. The network services require a descriptor which contains the details of all the VNFs, virtual links and forwarding graph of VNFs. Each of the VNFs contains it's own Virtual Network Function Descriptor (VNFD) and the information about the number of virtual machines it requires. Different MANO frameworks have their own descriptor schemata pertaining to the standard defined by ETSI. This framework-specific Network Service Descriptor (NSD) hinders the orchestration and management of VNFs between different MANO frameworks.

Firstly, the project aims at tackling the above mentioned problem with the implementation of translator and splitter engines, which would help translate the NSD and divide the VNFs to be deployed on different MANO frameworks, thus creating a framework-independent network service chain. Secondly, the project aims at the implementation of a MANO adaptor, that allows interaction between different MANO frameworks, exposes the network service instantiation interfaces of the underlying MANO frameworks and retrieves monitoring information about the network service status. The adaptor will mainly address MANO scalability challenges and perform state management. Lastly, the project aims at integrating these individual modules in order to provide an end-to-end network service delivery across different MANO frameworks.

2 Goals and Use Cases

In this chapter, the intended goals of the project and some use cases are being discussed.

2.1 Goals

The goal of the project is to develop a software suite which facilitates inter-operability between MANO frameworks, thereby enabling the management of a network service across a multi-vendor environment. To achieve this, the software suite is divided into 3 individual work packages (WPs), which are initially developed in parallel and are merged later. In the following sections, the individual goals of these work packages are explained in detail.

2.1.1 Service Descriptor Translator (SDT)

In this work package, a translator engine for a NSD is implemented. The Network Function Virtualization Orchestrator (NFVO) is the main orchestration unit of a MANO framework that manages the lifecycle of a network service. One of the main functions of a NFVO is registering a network service, by on-boarding it's NSD to the network service catalog. In a scenario where a network service is to be deployed among two different MANO frameworks, having their own respective NSD schema, SDT engine would help in translating a NSD schema from one MANO framework to the other framework.

2.1.2 Service Descriptor Splitter (SDS)

In the production environment, a network service is deployed over a vast geographical region spanning multiple domains. In such scenarios, there is a need to split a network service into smaller network services and deploy it over multiple domains. In this work package, a Service Descriptor Splitter (SDS) is implemented which splits the NSD of a network service. SDS takes NSD as an input that contains all the information elements which can be extracted to generate separate NSDs. In the proposed approach, the service graph is extracted from the input NSD and is split into subgraphs that result in a separate NSD which includes a set of elements such as VNFs, Virtual Links (VLs), forwarding graphs of VNFs etc, according to the specific MANO framework.

2.1.3 MANO Adaptor (MA)

A key component of a MANO framework is Virtualized Infrastructure Manager (VIM), which helps in assigning the hardware resources to virtual resources. The MANO framework has specific adaptors to communicate with their respective VIMs. For instance, Sonata (4.1.1) MANO framework has adaptors for OpenStack (4.2.1) and Kubernetes (4.2.2). However, when there is a spike in the service request beyond the capabilities of a single MANO instance, multiple MANO instances should be instantiated to balance the load. The goal of this work package is to build an adaptor that facilitates interaction between different instances of MANO frameworks, thereby exposing the underlying MANO framework's interfaces and enabling monitoring of the underlying service states.

Apart from the implementation of MA, the plan is to investigate MANO scalability challenges

2.1.4 Integration Of Work Packages

In this work package, the SDT, SDS, and MA are integrated. This integration will augment the functionality of the MA in terms of addressing the scalability challenges between instances of different MANO frameworks. It also extends the capabilities of both the SDS and SDT to split a NSD and then deploy a part of it on a different MANO framework. All these individual modules are planned to be implemented as microservices and are integrated keeping their individuality intact.

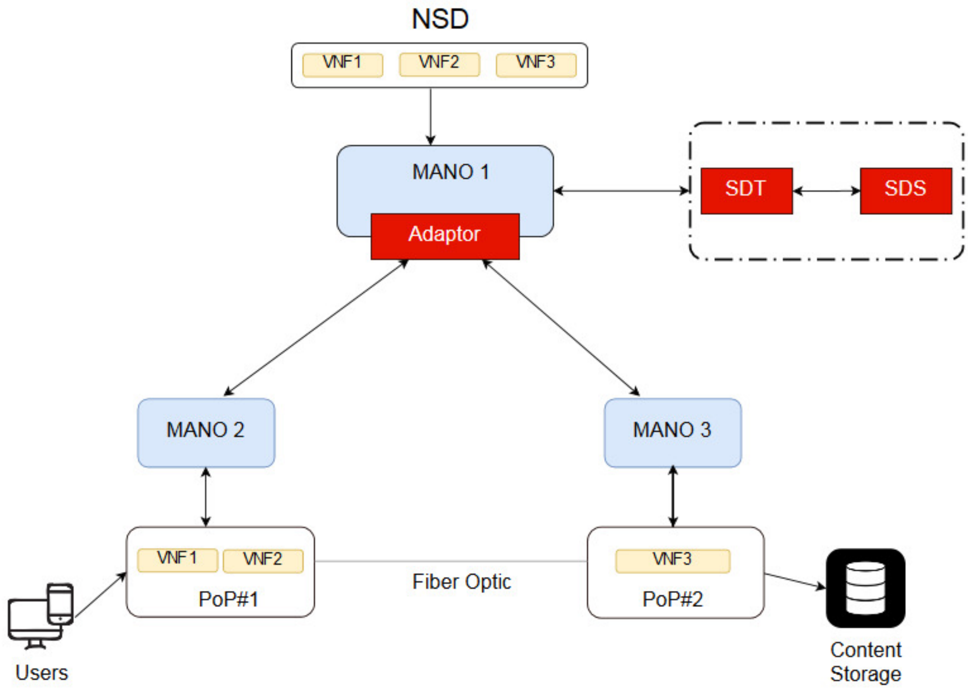

Figure 2.1: This figure visualizes the structure of the software suite.

2.2 Use Cases

2.2.1 Cross-MANO Framework Interaction

The MANO frameworks used by every network service provider varies from one another. The software suite with the help of SDT and SDS enables the deployment of network services across different frameworks.

For instance: Consider two network service Operators using different MANO frameworks. One of them uses Sonata framework [DKP+17] and another operator uses OSM framework [Ers13]. These frameworks have different NSD schemata(refer 4.1.1 and 4.1.2). NSD schemata contain VNFs, virtual links, and VNF forwarding graphs and also describes the deployment of a network service. By using a translator and splitter, these NSD schemata can be translated and split into a framework-specific schema. With this, operators can deploy and manage network services across different MANO implementations.

2.2.2 Hierarchical Orchestration

By using MA, dynamic instantiation of multiple MANO instances and inter-operability between different MANO frameworks can be achieved. The operator will be able to handle the resources in an efficient manner, as one MANO framework can manage a limited number of service requests, operators can explore options to include additional MANO instances under the existing MANO instance to mitigate the traffic load on a single instance. The resources can be provisioned based on the number of requests. This helps the operator in extending their profitability.

Actors : The Network Service Providers who would use features of SCrAMbLE.

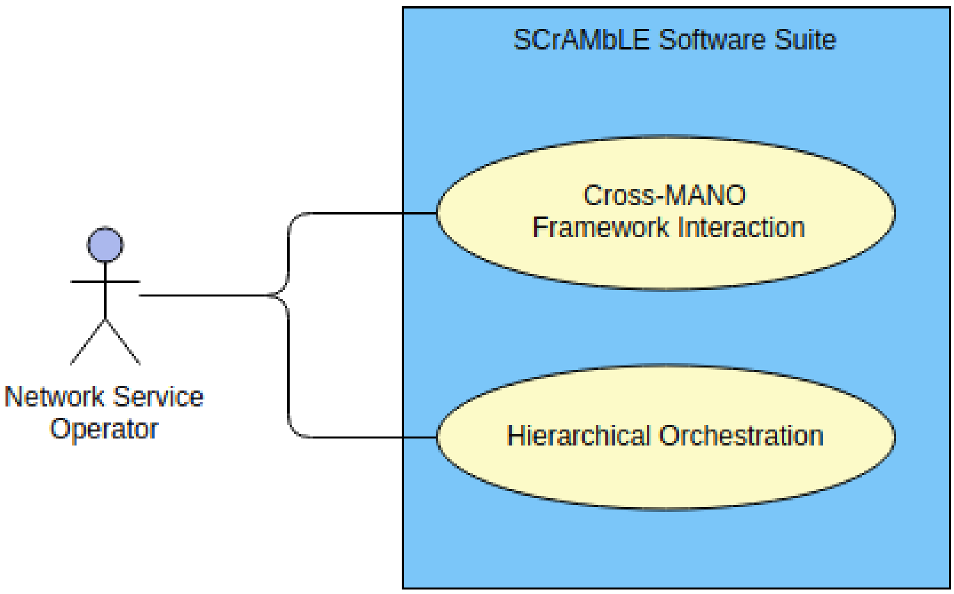

Figure 2.2: Use Case Diagram

3 Non-Functional Requirements

The important non-functional requirements for the work packages are defined here. Below is the overview of some requirements and description of their relevance in this project.

1. ***Scalability:*** One of the main requirements of any cloud-based networking, is its ability to adapt its throughput to varying load. Here, the load of particular network functions, load on the translator and splitter is also considered as an add-on to the overall load of the system. One of the main work packages mentioned in this project is the scalability support for the MANO framework. The adapter plays a major role in scaling up the defined system.

2. ***Reliability:*** The requirement of reliability aims at the ability of the system to provide continuous correct service. It is the continuation of the service in compliance with the service specification. It is one of the weak requirements that have to be met in any case. In this project, this can also be viewed as providing service for the required duration and then terminating.

3. ***Flexibility:*** Flexibility abstracts the ability to change the system according to changing requirements. Here in this project, the software suite should support new MANO frameworks with minimal changes.

4. ***Performance:*** This term summarizes many possible aspects like response time and Quality of Service (QoS). In this project, the software suite and its individual components have to be efficient without adding significant overhead to the MANO framework.

4 Technologies

4.1 MANO Frameworks

In this section, a few MANO frameworks and their NSD schemata are listed(list of a few parameters and their description). Among several frameworks, the plan is to select a few and set them up locally, deploy network services and support them in the project. This list is subjected to future additions or removals.

4.1.1 SONATA

SONATA is one of the NSO research projects which targets two crucial technological challenges in the foreseeable future of 5G networks as follows[SON].

- Flexible programmability
- Deployment optimization of software networks for complex services /applications.

SONATA complies with the ETSI NFV-MANO reference architecture. Following tables lists the important parameters of NSD [Docb].

Network Service Descriptor Section

Parameter	Description
Vendor	Identifies the network service uniquely across all network service vendors.
Name	Name of the network service without its version.
Version	Names the version of the NSD.
Author	It's an optional parameter. It describes the author of NSD
Description	It's an optional parameter, provides an arbitrary description of the network service.

Table 4.1: SONATA: Network Service Descriptor Section

Parameter	**Description**
network_functions	Contains all the VNFs that are handled by the network service.
Vnf_id	Represents a unique identifier within the scope of the NSD
Vnf_vendor	As part of the primary key, the vendor parameter identifies the VNF Descriptor
Vnf_name	As part of the primary key, the name parameter identifies the VNF Descriptor
Vnf_version	As part of the primary key, the version parameter identifies the VNF Descriptor
Vnf_description	It's an optional parameter, a human-readable description of the VNF

Table 4.2: SONATA: Network Functions Section

Network Functions Section

4.1.2 Open Source MANO(OSM)

OSM is an open source management and orchestration stack in compliant with ETSI NFV information models. OSM architecture splits between resource orchestrators and service orchestrators [dSPR+18]. Table 4.3 lists the parameters of NSD schema. [Doca]

Parameter	**Description**
id	Identifier for the NSD
name	NSD name
Short-name	Short name which appears on the UI
vendor	Vendor of the NSD
logo	File path for the vendor specific logo. For example, icons/mylogo.png. The logo should be a part of the network service.
description	Description of the NSD
version	Version of the NSD

Table 4.3: OSM: Network Service Descriptor

4.2 Virtualized Infrastructure Manager (VIM)

VIM is one of the three functional blocks specified in the Network Functions Virtualization Management and Orchestration (NFV-MANO) architecture. VIM is responsible for controlling and managing the NFV Infrastructure (NFVI), by provisioning and optimizing the allocation of physical resources to the virtual resources in the NFVI. Performance and error monitoring is also a key role of the VIM. Popular VIMs are discussed in the following sections.

4.2.1 OpenStack

OpenStack[1] is a community-driven open source cloud resource management platform. Compute, storage, and networking resources in a data center can be provisioned using Application Program Interfaces (APIs) or web dashboard provided by OpenStack component, for instance, NOVA is a component which can provide access to compute resources, such as virtual machines and containers. Network management is enabled by NEUTRON component which handles the creation and management of a virtual networking infrastructure like switches and routers. SWIFT component provides a storage system. By making use of many such components, OpenStack can deliver complex services by utilizing an underlying pool of resources.

4.2.2 Kubernetes

Kubernetes[2] (K8s) is an open-source platform for automation and management of containerized services, it manages computing, networking, and storage infrastructure. Kubernetes was initially developed by Google and now under Cloud Native Computing Foundation. Kubernetes Architecture consists of (1) Master server components – it is the control plane of the cluster and act as the gateway for administrators (2) Node Server Components – servers which are performing work by using containers that are called nodes, they communicate with the master component for instructions to run the workload assigned to them. Kubernetes provides comprehensive APIs which are used to communicate between the components and with the external user.

[1] https://www.openstack.org/
[2] https://kubernetes.io/

5 Related Work

In this chapter, the relevant research efforts that can be used to achieve the goals are discussed. Firstly, the standards and specifications for orchestration and management of NFV in the section are discussed 5.1. The fundamental aspect of a service deployment is the NSD, in the section 5.2 the trends and options of NSDs and research papers that try to mitigate the interoperability challenges between different MANO frameworks are discussed. Section 5.3 will be a brief account of the MANO scalability problem.

As this is the initial project proposal, the state-of-the-art could change progressively and the approach will be updated accordingly.

5.1 Standards and Specifications

SDN decouples network control from forwarding with programmable ability. With the decoupling and programmability, SDN brings many benefits such as efficient configuration, improved performance, higher flexibility [XWF$^+$15]. ETSI NFV [NFV2] architecture virtualizes network functions and enables dynamic and flexible selection of service functions. In ETSI NFV architecture, Network Function Forwarding Graph (VNF-FG), which consists of multiple network functions, is defined to describe network service. Internet Engineering Task Force (IETF) Service Function Chaining (SFC) working group also proposes the SFC architecture in RFC 7665 [HP15]. An SFC defines an ordered set of network Service Functions (SFs) for delivery of end-to-end services. Reference [QE16] designs a protocol named Network Service Header (NSH) to decouple the service from topology. An intelligent control plane is proposed to construct service function chains but does not consider the multi-domain situation [B$^+$16].

ETSI NFV designs a basic frame for NFV-MANO. It defines VIM, VNF Manager (VNFM) and NFV Orchestrator (NFVO) for management and orchestration of Network Functions Virtualization Infrastructure (NFVI),VNF and network services [ETS14].

5.2 Network Service Description and Interoperability

The description of the network service plays an important role in integration and interoperability of different MANO frameworks. According to ETSI, network service is the "composition of network functions and defined by its functional and behavioral specification." Following this approach, a network service can be defined as a set of VNFs and/or Physical Network Functions (PNFs), with virtual links (VLs) interconnecting them and one or more virtualized network function forwarding graphs(VNFFGs) describing the topology of the network service.

Garay et al. [GMUJ] emphasize on NSD, required to allow the different components to inter-operate by comparing the NSD templates by OpenStack (HOT [1]) and OASIS (TOSCA[2]). A strawman model is proposed in the paper to address the upcoming interoperating challenges. The aim is to build a mechanism to translate the NSDs in order to facilitate the interoperability between different MANO frameworks.

5.3 Scalability and Hierarchical Orchestration

MANO framework faces significant scalability challenges in large-scale deployments. The amount of infrastructure a single instance of MANO framework can manage is limited. Network service scaling with NFV is discussed in paper [AHOLA$^+$18]. It also shows different procedures that the Network Function Virtualization Orchestrator (NFVO) may trigger to scale a network service according to ETSI specifications and how NFVO might automate them. Abu-Lebdeh et al. [ALNGT] explores the effects of placement of MANO on the system performance, scalability and conclude by suggesting hierarchical orchestration architecture to optimize them. They formally define the scalability problem as an integer linear programming and propose a two-step placement algorithm. A horizontal-based multi-domain orchestration framework for Md-SFC(Multi-domain Service Function Chain) in SDN/NFV-enabled satellite and terrestrial networks is proposed in [LZF$^+$]. The authors here address the hierarchical challenges with a distributed approach to calculate the shortest end-to-end inter-domain path.

The main intention is to answer the MANO scalability challenges, by exploring the optimal number of MANO deployments in a system and optimal hierarchical level. Also, how to manage the state of such a system dynamically.

[1] Heat orchestration template (HOT) specification:
http://docs.openstack.org/heat/rocky/template_guide/hot_spec.html

[2] Topology and Orchestration Specification for Cloud Applications (2013):
http://docs:oasis-open:org/tosca/TOSCA/v1:0/os/TOSCA-v1.0-os.html

6 Work Packages

6.1 Translator

In a hierarchical architecture involving different MANOs, there is a need of conversion of network descriptors to schemas of respective MANO. Service Descriptor Translator (SDT) serves the purpose of translating network descriptors, namely NSDs and VNFDs from schema of SONATA Pishahang to that of OSM and vice versa.

In a scenario, where a parent MANO, say Pishahang decides to deploy one of the network services in its lower hierarchy MANO, say OSM, the NSD and VNFD(s) need to be converted to the descriptor schema of OSM. In such an event, the Scramble plugin calls the translator service and sends the descriptors to the SDT, where the translation of the descriptors takes place and the translated descriptors are sent to Adaptor utility for deployment in appropriate MANO. Figure 6.1 gives a high level view of Translator.

Scramble plug-in installed within the parent MANO forwards the network descriptors with service request to Scramble Main-Engine. Main-Engine checks the service request and sends the network descriptors to Translator along with the information of destination schema. On receiving the network descriptors, SDT translates the same to requested schema and calls the validator function to validate the translated network descriptors. Once validation is complete, the translated descriptors will be sent to Adaptor for deployment.

6.1.1 Architecture & Work flow

6.1.2 usage

+ code snippets and explanations

6.1.3 challenges

6.1.4 Future scope of this work package

6.2 Splitter

Communication between MANO frameworks and Scramble happens through plug-ins. Plug-ins will have some random logic to generate a multiple set of Network Functions into which a

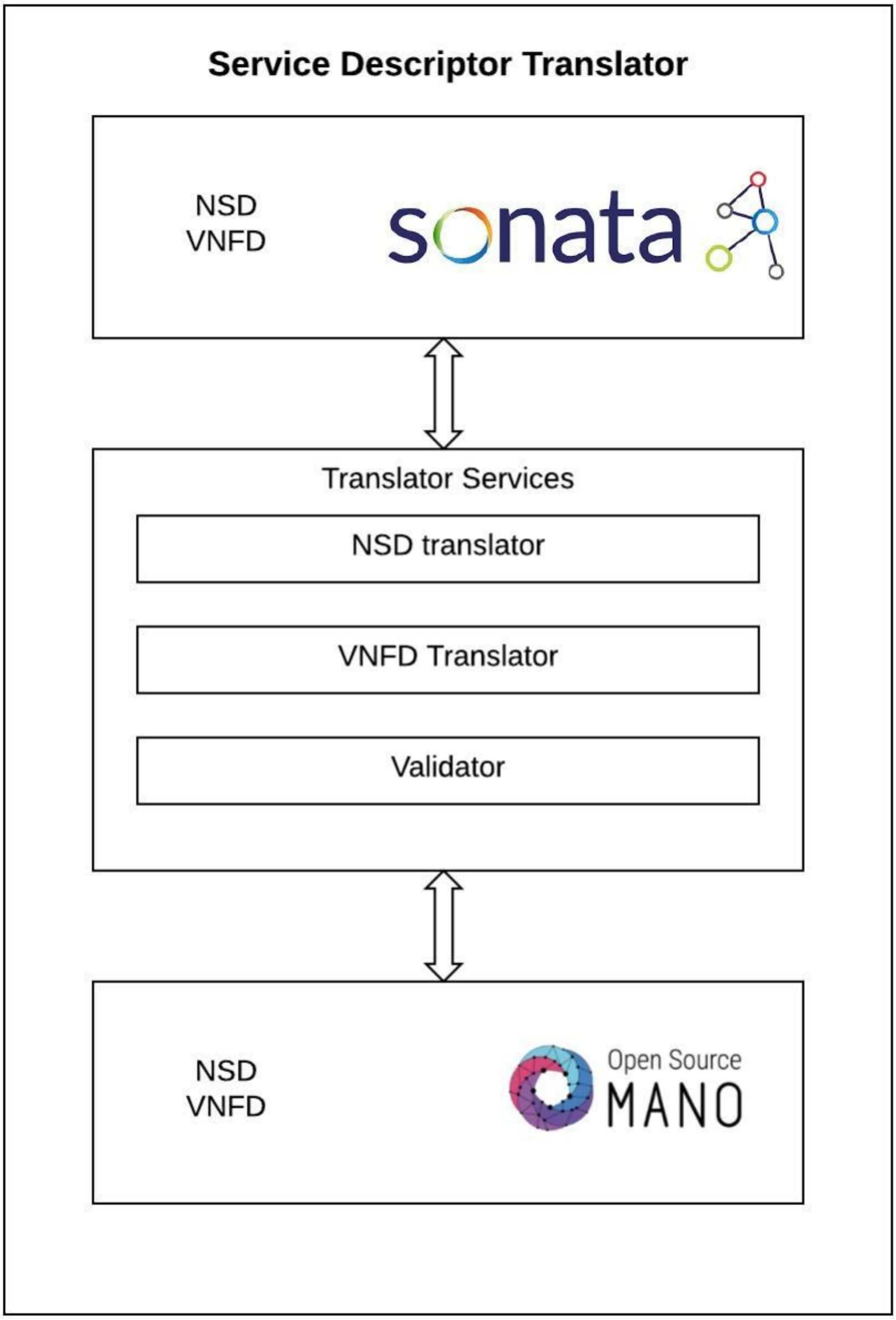
Service Descriptor Translator
NSD
VNFD
sonata
Translator Services
NSD translator
VNFD Translator
Validator
NSD
VNFD
Open Source
MANO

Figure 6.1

Network Service needs to be divided. The multiple set of NFs is then forwarded to Gateway micro-service along with the NS file which just receives the request and forward it to the Main-Engine. The Main Engine acts as an interface to all three utilities including Splitter. Main engine is responsible for interoperability and communication between all utilities. All three utilities are containerized using docker for easy distribution and scaling. Once the Splitter receives the request for splitting along with the parameters it splits the NSD into sub NSDs.

6.2.1 Architecture & Work flow

Before actual splitting of NSD, Splitter first validates if the the multiple set it received is valid or not. Following things are validated by the splitter.

- The total number of NFs mentioned in all the set matches the number of NFs defined in the NSD.
- There are no invalid NFs in the sets received.

After validation the actual splitting starts. We have created classes for different sections of a NSD which encapsulate all the attributes and its values into a single unit which makes it very easy to process. Once the objects are set they are passed to different splitting functions based on there type. We have two different processing units for OSM and SONATA. Following are some functions responsible for splitting the NSD.

- **Set General information:** This function copies all the general information from the main NSD to the sub NSDs. Information includes Vendor, author, Version, Descripton etc.
- **Split Network Functions:** This function splits the Network functions from NSD to sub sets according to the request parameter received.
- **Split Virtual Links:** When a NSD is splitted into different parts, its topology changes. Change in topology results in changing of Virtual Links. For example if A, B and C are three Nfs and we are splitting them in such a way so that A and B remain in one NSD and C in separate NSD. A virtual link between B and C now does not make sense. So this link should be broken down and B's output should be connected to the external end point which was connected to C's input earlier. This function splits these kind of Virtual Links.
- **Split Forwarding Graph:** As explained in the above section, once the topology changes, the respective Forwarding graph also changes. Split forwarding graph pulls out the set of connection points and newly created virtual links and sets them in the sub NSDs.

Once the Splitting is done, create file is responsible for creating YAML files depending on the number of sub NSDs created. These files are saved in the file system which can be downloaded or moved forward to the adopter for deployment purpose. Following figure 6.2 graphically represents the splitting architecture.

6.2.2 usage

+ code snippets and explanations

6.2.3 challenges

6.2.4 Future scope of this work package

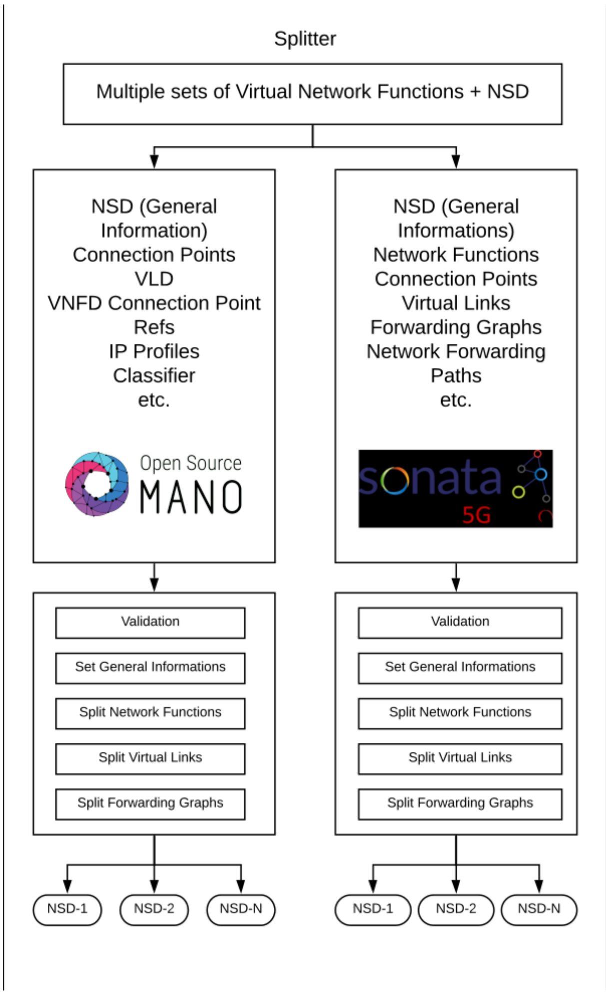

Figure 6.2: Scramble Architecture

6.3 Adaptor

Facilitating easy communication between MANOs is an important aspect of scramble. Adaptor is a component that enables communication between MANOs by wrapping the REST APIs of MANOs in python code.

Python MANO Wrappers (PMW) is a uniform python wrapper library for various implementations of NFV Management and Network Orchestration (MANO) REST APIs. PMW is intended to ease the communication between python and MANO by providing a unified, convenient and standards oriented access to MANO API.

To achieve this, PMW follows the conventions from the ETSI GS NFV-SOL 005 (SOL005) RESTful protocols specification. This makes it easy to follow and the developers can use similar processes when communicating with a variety of MANO implementations.

PMW is easy to install, use and well documented. Code usage examples are available along with the detailed documentation at the following link `https://python-mano-wrappers.readthedocs.io/en/adaptor/`.

PMW is planned and released as an independent library. In scramble, PWM helps in inter communication of different instances of MANO, thereby creating opportunity for more advanced feature set, for example, hierarchical scaling. Operations such as on-boarding of NSD and VNFD, instantiation and termination of NS can be performed with ease.

6.3.1 Architecture & Work flow

Standards based approach is a fundamental design principle behind PMW's design. A Common interface template is defined in compliance with SOL005 which contains the blueprint for all the methods mentioned in the standards. These methods are divided into different sections as per SOL005 into the following:

- **auth:** Authorization API
- **nsd:** NSD Management API
- **nsfm:** NS Fault Management API
- **nslcm:** Lifecycle Management API
- **nspm:** NS Performance Management API
- **vnfpkgm:** VNF Package Management API

In the figure 6.3, different sections of PMW are visualized. As part of the scramble project, support for Open Source MANO (OSM) and Sonata was implemented based on the common interface. This is represented by the dotted lines to OSM and Sonata modules. These modules are based on the common interface and implement the methods it has defined. Wrappers also support additional functionalities of Pishahang, which is an extension of Sonata.

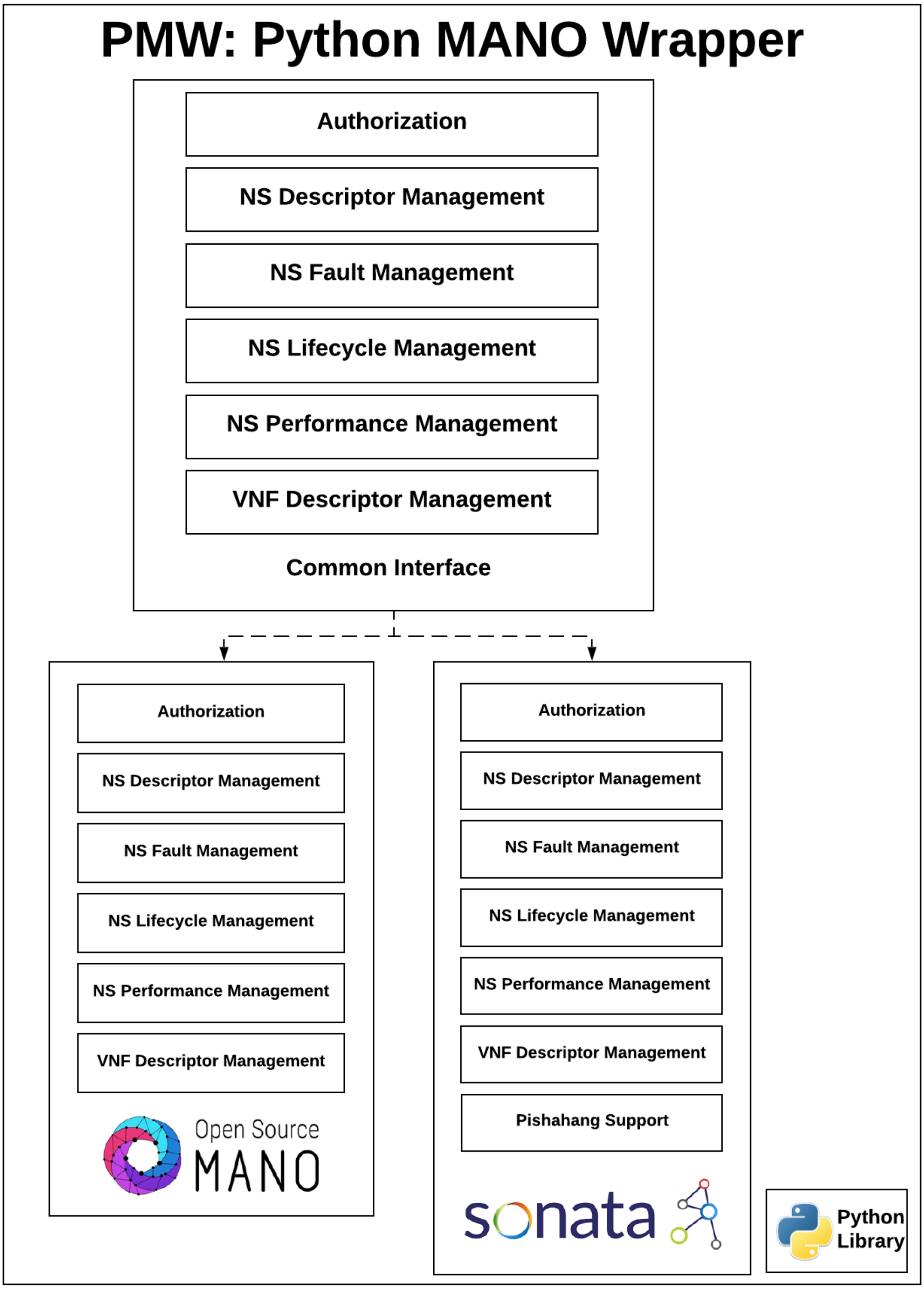

Figure 6.3: PWM Common interfaces

6.3.2 Installation and usage

PWM can be installed using pip by using this command `pip install python-mano-wrappers`.

A simple script to get started with PWM is shown in the Listing 6.1, here, the wrappers are imported and a client object is created according to the MANO type. Currently supported are OSM and Sonata. Such a client object can be used to make REST calls relevant to the MANO type. An example usage to retrieve all the network service descriptors of OSM can be seen from the listing 6.2, here, the OSMClient module is used to first fetch an auth token and further using the auth token to fetch the relevant information, in this case NSD descriptors.

```python
import wrappers

username = "admin"
password = "admin"
mano = "osm"
# mano = "sonata"
host = "osmmanodemo.com"

if mano == "osm":
    _client = wrappers.OSMClient.Auth(host)
elif mano == "sonata":
    _client = wrappers.SONATAClient.Auth(host)

response = _client.auth(
                username=username, password=password)

print(response)
```

Listing 6.1: Simple wrapper code to fetch token

```python
import wrappers

osm_nsd = wrappers.OSMClient.Nsd(HOST_URL)
osm_auth = wrappers.OSMClient.Auth(HOST_URL)

_token = json.loads(osm_auth.auth(
                    username=USERNAME,
                    password=PASSWORD))

_token = json.loads(_token["data"])

response = json.loads(osm_nsd.get_ns_descriptors(
                        token=_token["id"]))
response = json.loads(response["data"])
```

Listing 6.2: Code to fetch all NSDs in OSM

6.3.3 challenges

Implementing such a python wrapper for a REST API is straight forward from the implementation perspective. However, the challenges that we faced are when identifying the required functional documentation from the respective MANOs. OSM and Sonata do not yet fully support the ETSI suggested endpoints and this combined with the lack of unified documentation, made it difficult in the begining to decide on the scope of supported functionalities.

6.3.4 Future scope of wrappers

PWM is built with easy maintanability and feature addition in mind. PWM makes it easy to add support for other MANOs. We expect MANO developers to use the common interface that we have suggested to add support to their REST APIs in python.

7 Scramble architecture

Scramble is a tool to realize scalability with hierarchical orchestration. While the higher level orchestrator (Parent MANO) still controls the life-cycle of the deployed network services, scramble acts as a bridge between parent MANO and lower level orchestrator (child MANO) thus allowing interconnection between different MANO's to provide end-to-end service.

7.1 Architecture

The services of scramble resides as a plugin within MANO frameworks such as SOTANA/Pishahang and Open Source MANO (OSM) and enables cross MANO communication. Service requests received by the higher level MANOs in the hierarchy can be routed to lower level MANOs using scramble plugin based on monitoring parameters.

Each child MANO's in-turn contains scramble plugin which allows multiple other MANO's to be added as a child hence achieving hierarchical orchestration (See Figure 7.1).

Scramble is composed of three main services.

- Translator
- Splitter
- Wrapper(adaptor)

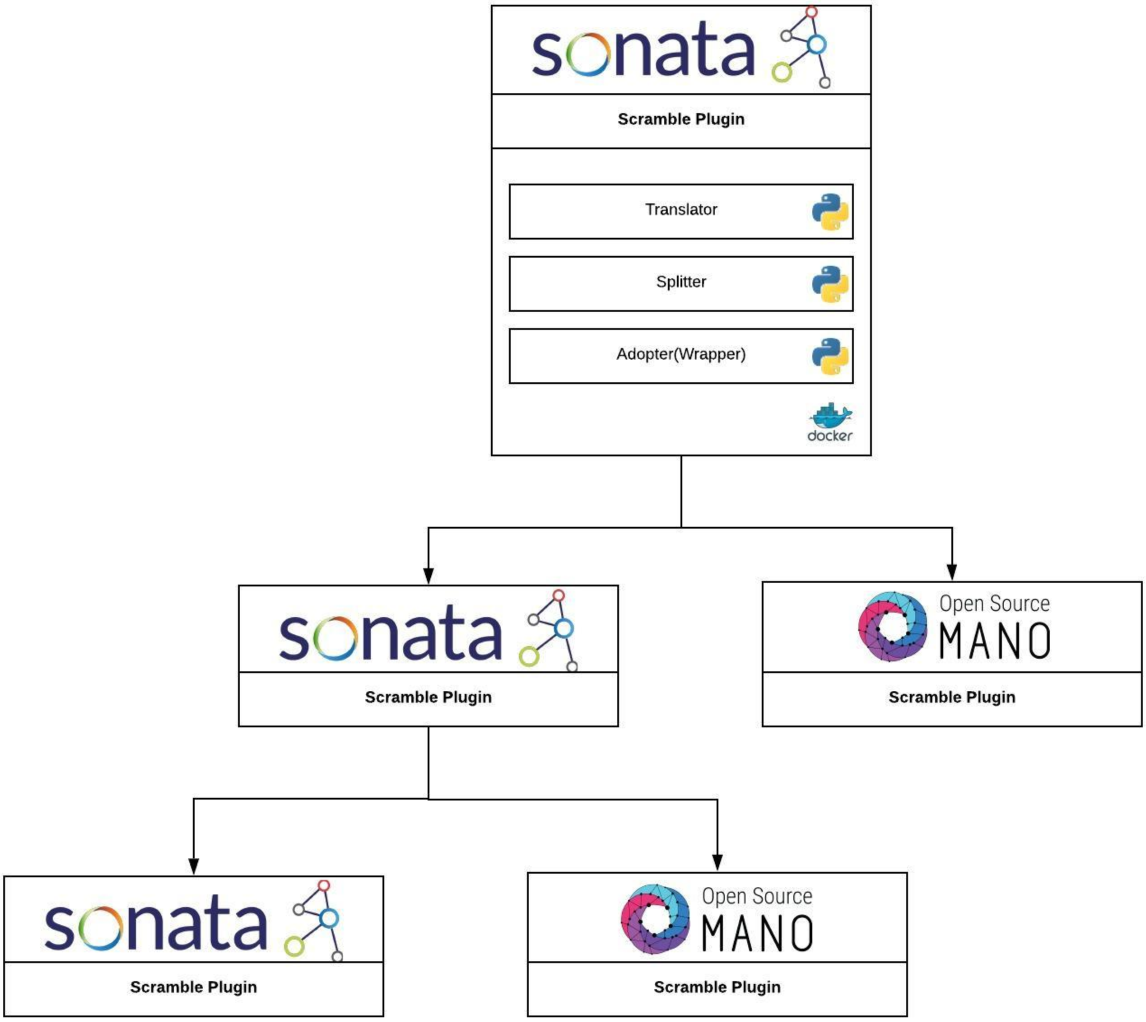

Figure 7.1: Scramble Architecture

8 MANO Scalability

In this chapter we discuss the two directions we explored to investigate MANO orchestrator scalability. First, we discuss the scalability plugin that was added to pishahang. The scalability plugin adds 3 main functionalities to pishahang, 1) spawn new child instances of pishahang by allocating new physical resources, 2) Redirecting requests from parent MANO to the child instances and 3) managing the state of child instances. Second, we discuss the experiments conducted on OSM and Pishahang to understand the resource utilization. We also propose a more generic framework to characterize and analyze MANO under load.

8.1 Introduction

Add content

8.1.1 Definition of scaling

'Scalability' is defined in different ways in various academic work. Some of the definitions are listed below.

- "The ability of a particular system to fit a problem as the scope of that problem increases (number of elements or objects, growing volumes of work and/or being susceptible to enlargement)." [?]
- "Scalability of service is a desirable property of a service which provides an ability to handle growing amounts of service loads without suffering significant degradation in relevant quality attributes. The scalability enhanced by scalability assuring schemes such as adding various resources should be proportional to the cost to apply the schemes." [?]
- "Scalability is the ability of an application to be scaled up to meet demand through replication and distribution of requests across a pool or farm of servers." [?]
- "A system is said to be scalable if it can handle the addition of users and resources without suffering a noticeable loss of performance or increase in administrative complexity" [?]

8.1.2 Why does a MANO need scaling?

In recent years, distributed systems have gained an increase in the number of users and resources. Scaling such a system is an important aspect when large user requests have to be served without compromising system performance or increase in administrative complexity. In terms of MANO, when there are a large number Network Service(NS) instantiation of various network functions, they need to be instantiated considering all the relevant metrics of the system.

System load: In a distributed system, the system load is the large amount of data that is to be managed by network services increasing the total number of requests for service. The load on a MANO can be defined in terms of it's load on NFV Orchestrator (NFVO) to process large number of tasks like on-boarding, instantiation and monitoring of VNFs. The NFVO of a MANO receives monitoring information which also increases the load on NFVO triggering it to scale the network service across multiple MANOs in a distributed system [**?**].

Lifecycle Management & service provisioning: To provision a network service, the NFVO of a MANO's functionality include instantiation, global resource management scaling in/out , event correlation and termination of services. These functionalities form the lifecycle of NS. With the increase in instantiation of NS over a distributed network, the lifecycle management of each service is a overhead, hence increasing the provision time. This can be better handled when the MANO can be scaled out. To manage services with a closer proximity of geographical region, MANOs could be scaled in.

8.2 Scalability Plugin

8.2.1 Introduction

8.2.2 Architecture

8.2.3 Workflow

8.3 Experiments

In this section, only talk about the 180 experiment that gave te horizontal graphs

8.3.1 Idea

Here, talk about the idea behind running the experiment and getting to know the docker containers that took the most resources

8.3.2 Testbed

Explain about the machine configuration used to run the experiments, what OS, how many servers and what VIMs where installed, what is the configuration and how python-mano-wrappers were used to send the requests.

Basically explain the whole experiment setup that was used to conduct the experiment.

8.3.3 OSM Results

The 8.1, 8.2 and 8.3 shows the CPU utilization, memory utilization and CPU utilization throught the life cycle of the experiment respectively. Let us consider the important dockers in each case and list their functionalities to realize why they have consumed maximum CPU.

CPU utilization

The The 8.1 shows CPU utilization. The first 5 OSM dockers are:

- **osm_ro:** This is the Resource Orchestrator for OSM. The resource orchestrator is responsible for co-ordinating resource allocation across multiple geo-distributed VIM types. It is responsible in processing the resource-allocation requirements of the VNF as per parts of the VNFD and driving the VIM to allocate appropriate compute, network, and storage resources for the deployment of VNFs with their interconnection.

- **osm_lcm:** This is the Life Cycle Management module for OSM. This docker is responsible for set of operations related to the life cycle of a VNF and NS

  - *NS LCM operations:* 1) On-board Network Service 2) Instantiate Network Service 3) Scale Network Service 4) Update Network Service by supporting Network Service configuration changes Create, delete, query, and update of VNFFGs associated to a Network Service. 5) Terminate Network Services
  - *VNF LCM operations:* 1) Instantiate VNF (create a VNF using the VNF on-boarding artefacts) 2) Scale VNF (increase or reduce the capacity of the VNF). 3) Update and/or Upgrade VNF (support VNF software and/or configuration changes of various complexity). 4) Terminate VNF (release VNF-associated NFVI resources and return it to NFVI resource pool)

- **osm_mon:** This is the monitoring module for OSM. The main task of this docker is to retreive metrics from VIM. It keeps polling VIM every 30 seconds.

- **osm_ro_db:** This is called RO database module. RO module in OSM maintains its own database called ro-db. (RO module is usually edited to enable or disable this particular module) This module takes care of database related operations. According to the osm/ro.git, ro-db stores different IDs like vnfd id ,osm id, tenant id, vim id etc. There is something called scenario. ro db stores these scenarios and scenario ID and then a scenario gets executed when the scenario id matches with the right tenant id, osm id and vnfd id. It stores vim actions, wim actions and they are processed sequentially.

- **osm_nbi:** North Bound Interface of OSM. Restful server that follows ETSI SOL005 interface. It is the unique entry point for all the interactions with the OSS/UI system. This serves as the interface for MANO operations. (OSM's NBI offers all the necessary abstractions to allow clients for the complete control, operation and supervision od the NS.)

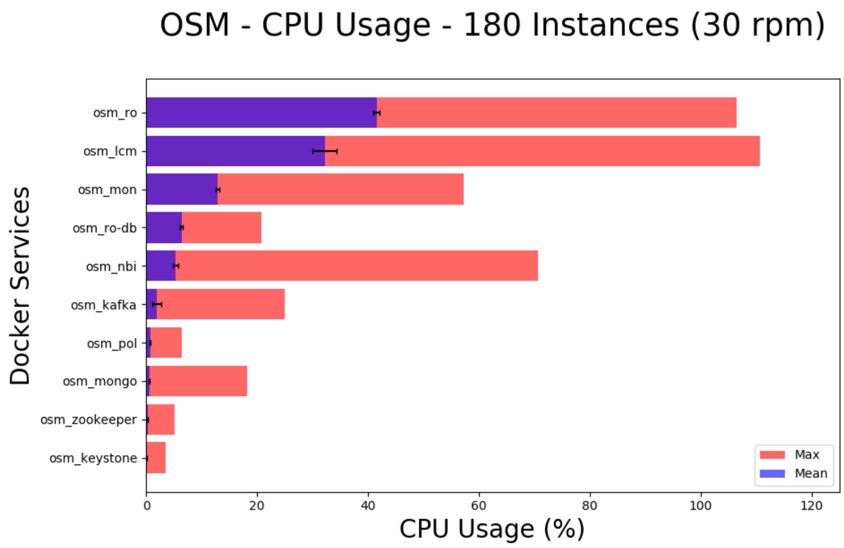

Figure 8.1: OSM CPU

Memory Utilization

The first 5 OSM dockers in memory utilization graph are:

- **osm_mongo:** This is a common non relational database for OSM modules.
- **osm_kafka:** This module provides a Kafka bus used for OSM communication.
- **osm_nbi:** This is the north bound interface for OSM module. The functionalities of this module remains the same(as explained in the previous subsection)
- **osm_light-ui:** This docker is an implementation of a web GUI to interact with the Northbound API. (The framework allows editing, validating, visualizing the descriptors of services and components both textually and graphically)
- **osm_ro_db:** This is the RO database. The functionalities of this module remains the same (as explained in the previous subsection)

Lifecycle

We now have the life cycle graphs of the entire experiment. This shows the distribution of the CPU usage among the top 3 dockers throughout the experiment. The experiment lasted for about 10 minutes.

We can observe that the ro module of OSM got continuous requests over the time and at the end all the termination requests came at once. Also LCM , mon modules continously processed the requests. mon module retrieved metric information continuously and the same trend remains throughout the experiment.

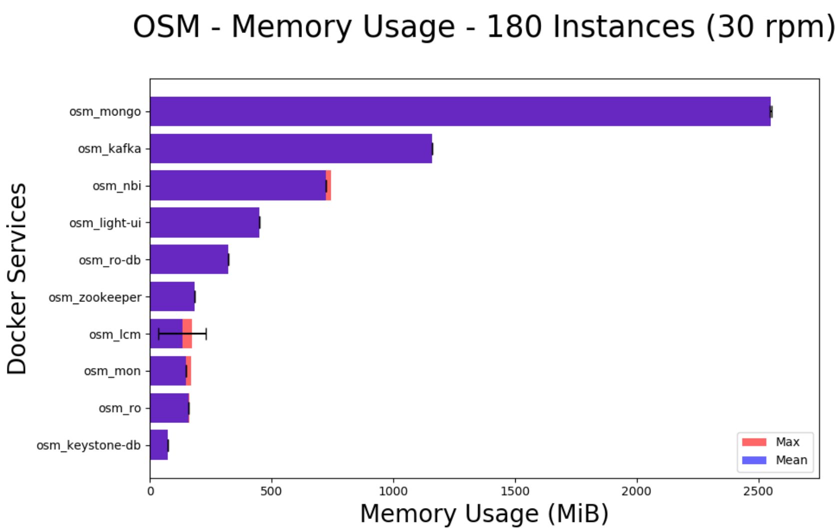

Figure 8.2: OSM MEM

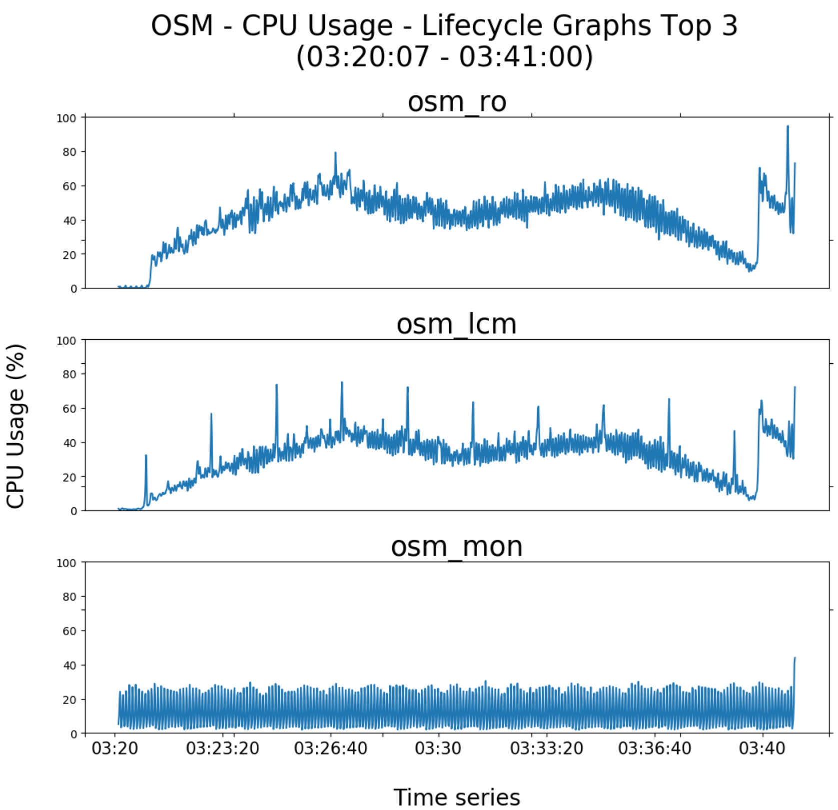

Figure 8.3: OSM LS

8.3.4 Pishahang Results

Pishahang Just like you did in the presentation, explain the functionalities of the top 5 dockers and why they are taking

CPU

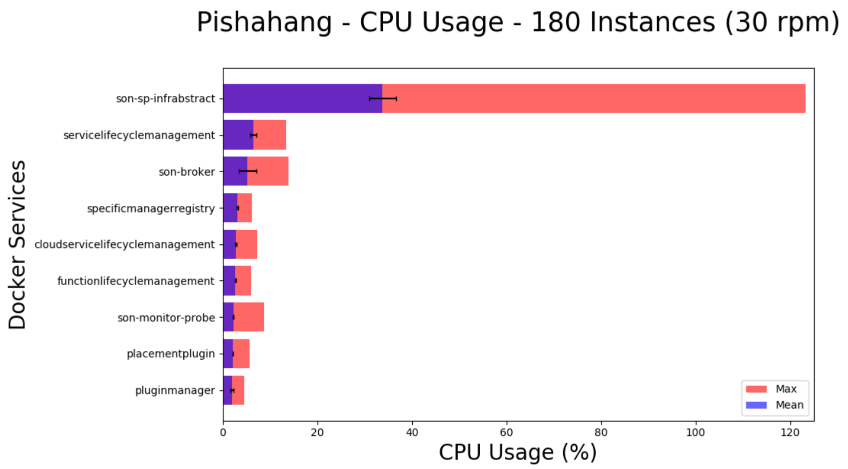

Figure 8.4: Pishahang CPU

Memory

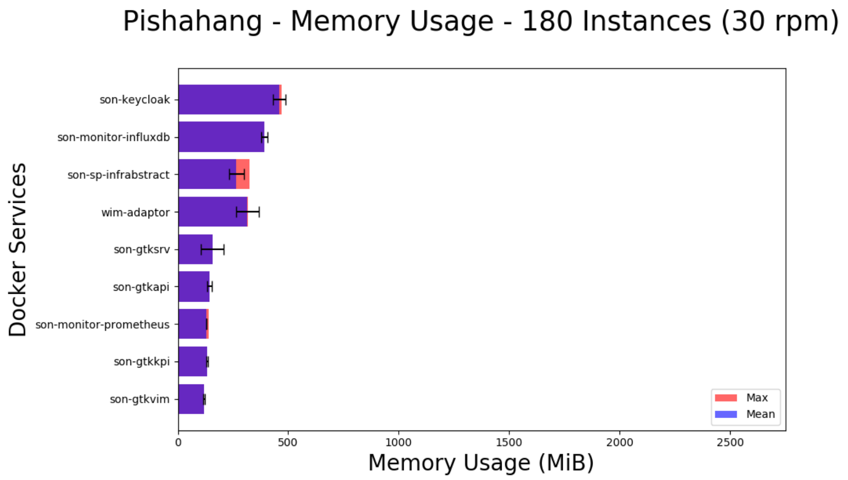

Figure 8.5: Pishahang MEM

Lifecycle

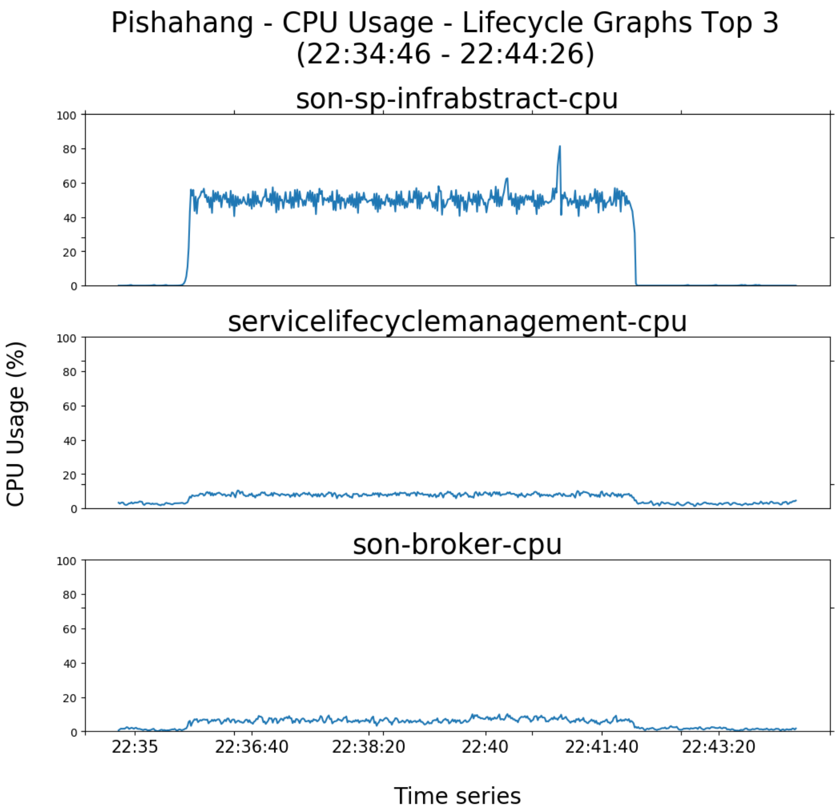

Figure 8.6: Pish LS

8.3.5 Summary of issues of the experiment

Blockers: VIM support issues, openstack not stable, k8 not supported in OSM. Why 180. RPM issues

8.3.6 Inference from the experiment

about identifying the top dockers, some more info about the lifecycle graphs

8.4 MANO Benchmarking Framework

8.4.1 Introduction

MANO Benchmarking Framework (MBF) is a result of a small script that was used to run the experiments discussed in the previous sections. The idea of MBF is to provide MANO developers with a generic framework for running experiments on MANO. MBF mainly provides the following 1) Easy interfacing with MANO instances by using python-mano-wrappers, 2) Ability to run experiments with different service descriptors, 3) Collection of performance metrics in convenient data format and 4) Flexible graphing mechanism of the collected data.

8.4.2 Design

MBF is designed for ease of use and low barrier to entry for developers. We explain the choice of tools that are used in MBF in the following list.

- **Netdata**[1] is the metrics monitoring system for MBF. Netdata captures relevant system metrics and provide powerful APIs to query the recorded data in a suitable format.
- **Python** as the choice of scripting language was obvious as the MANOs itself are implemented in python.
- **python-mano-wrappers** is used to provide access to REST APIs of MANOs from python.
- **Docker** is used to containerize MBF, thus making it easy to distribute and portable.
- **Matplotlib** is the graphing library for MBF due to its flexibility and ease of use.
- **Flask** as a python server that can be used to provide additional interactions with the experiment runner.

8.4.3 Parameters and KPIs

Parameters

MBF has experiment parameters that can be altered. The following paramenters are supported

- **Descriptors** NSDs and VNFDs can be changed. a list of NSDs/VNFDs is also supported. When a list of descriptors are provided, the experiment will be run for each of the descriptors.
- **Number of instances** Total number of instantiation requests to be sent to the MANO.
- **Number of runs** number of re-runs of the same experiment to be run. This is performed to measure the variance in results.
- **Requests per minute (RPM)** The rate at which the instantiation requests are sent to the MANO
- **Observation Time** The observation time after the instantiation requests are sent. This can be used to collect metrics post instantiation to observe how MANO behaves in the monitoring phase.

[1] https://github.com/netdata/netdata

- **Inter-experiment Delay** is the time between experiment runs. This is altered to give enough time for the VIM to terminate and cleanup instances from the previous experiment runs if any.
- **Skip experiment on error** if set to true, the current run is skipped due to a failed instance on the VIM.

8.4.4 Key Performance Indicators

MBF stores resource utilization metrics during the experiment and generates graphs to visualize the results. However, these are only examples and the further possibilities are supported by the framework. The metrics are stored as CSV files.

- **CPU** Overall system CPU usage is recorded as well as the individual docker micro service CPU usage metrics are stored.
- **Memory** Overall system memory usage along with the individual docker micro service memory usage is stored.
- **System Load** The 1m, 5m and 15m moving averages of system load values provided by the linux kernel is stored.
- **Status Tracking** The status of all instances are stored by polling the VIM every 5 seconds over the experiment lifetime. This enables to track the status change over time.
- **End-to-end Deployment Time** is the time elapsed to deploy all the instances on the VIM.
- **Individual Deployment Time** is the time taken by each instance for deployment. This is also split into time taken by MANO and VIM.

8.4.5 Steps for the automated experiment run

A walkthrough of the experiment runner

8.4.6 Example Use Cases

Explain results acquired from MANO benchmarking tool

Comparison of different network services

Comparison of different network services

Container vs VM Orchestration

Container vs VM Orchestration

Scaling Plugin Evaluation

8.4.7 Future scope

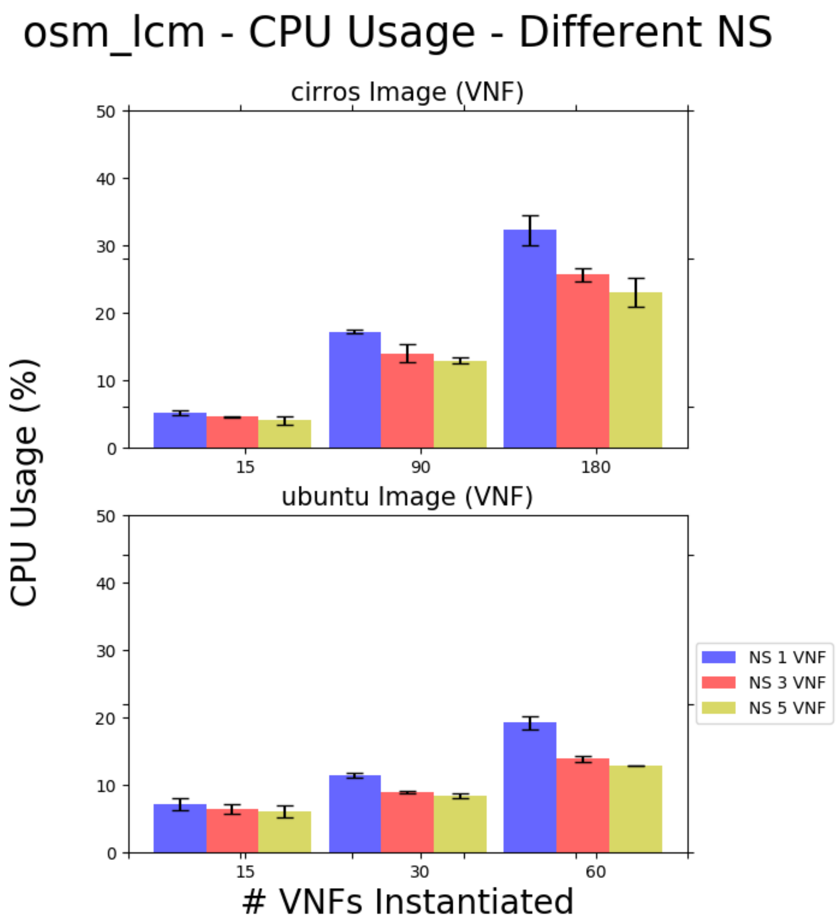
osm_lcm - CPU Usage - Different NS
cirros Image (VNF)
ubuntu Image (VNF)
CPU Usage (%)
VNFs Instantiated
0
10
20
30
40
50
15
90
180
30
60
NS 1 VNF
NS 3 VNF
NS 5 VNF

Figure 8.7

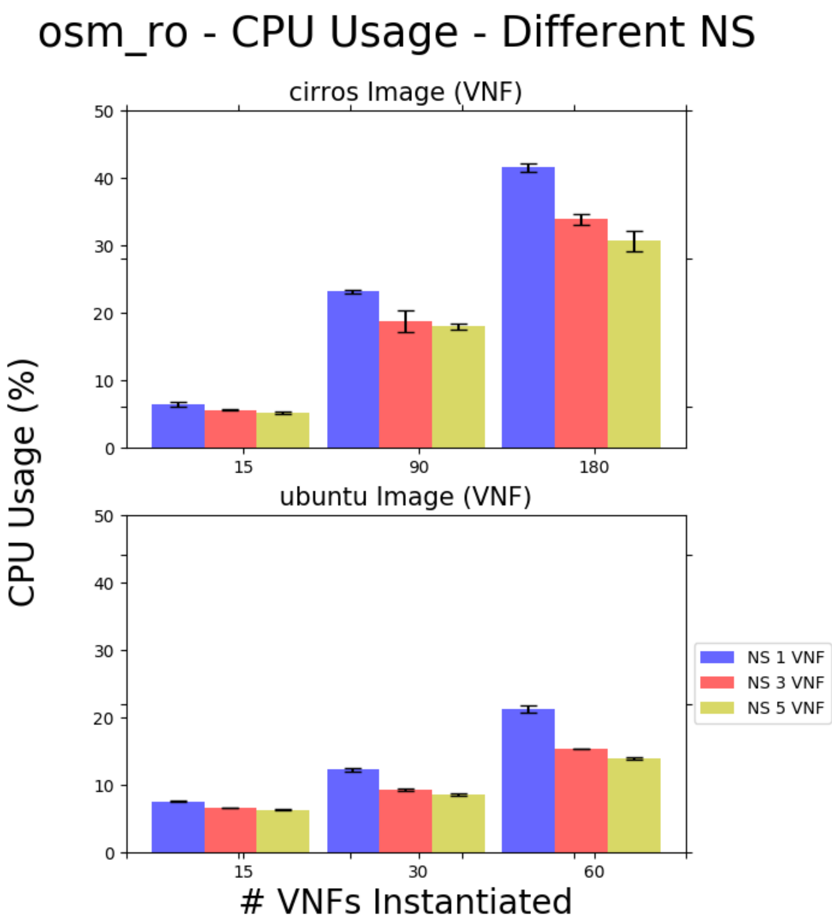
osm_ro - CPU Usage - Different NS
cirros Image (VNF)
ubuntu Image (VNF)
CPU Usage (%)
VNFs Instantiated
0
10
20
30
40
50
15
90
180
30
60
NS 1 VNF
NS 3 VNF
NS 5 VNF

Figure 8.8

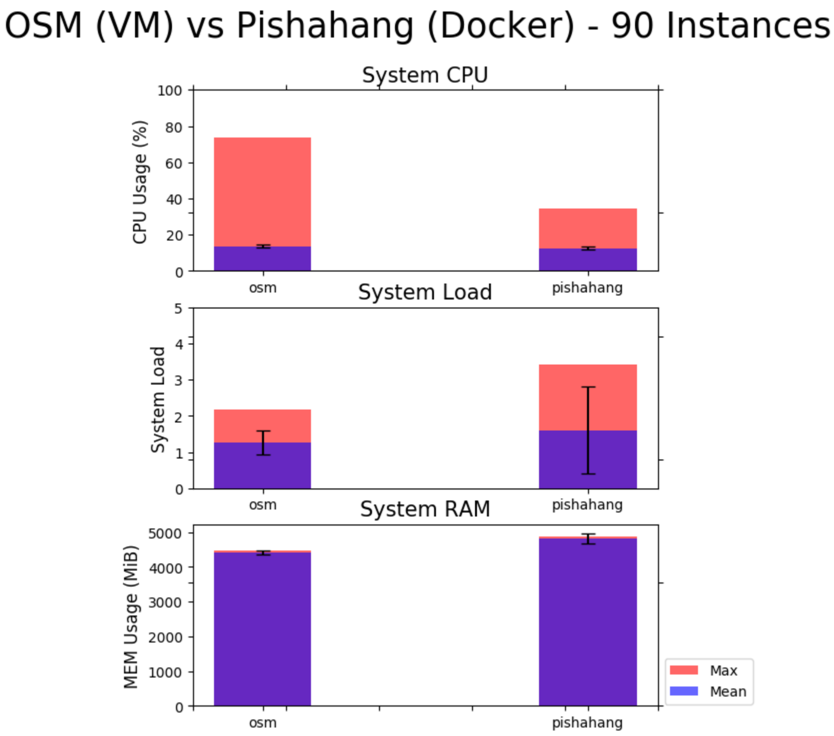

Figure 8.9

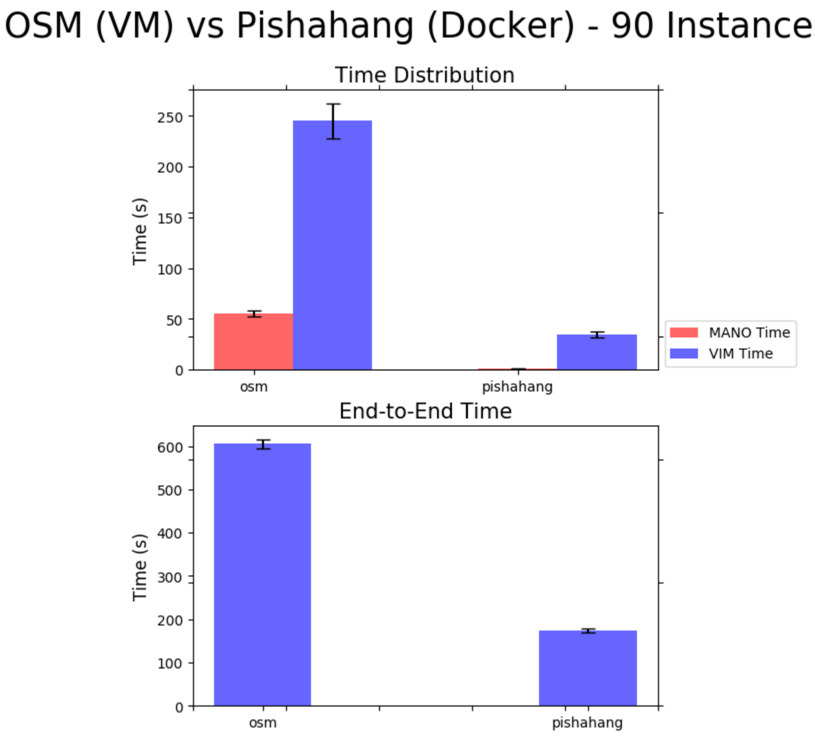

Figure 8.10

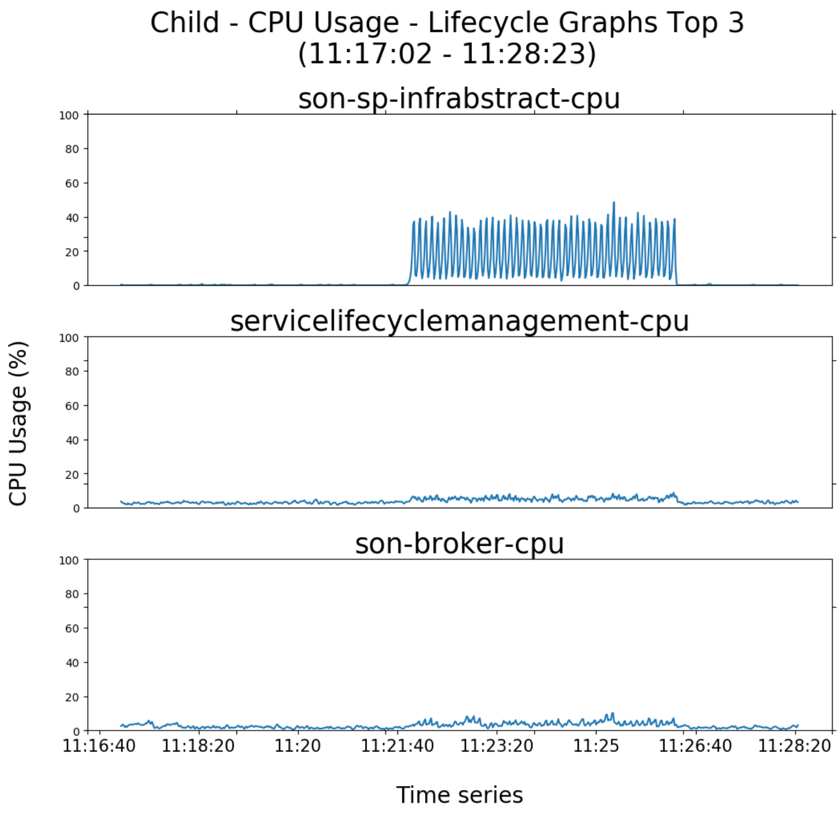

Figure 8.11: Child scaling

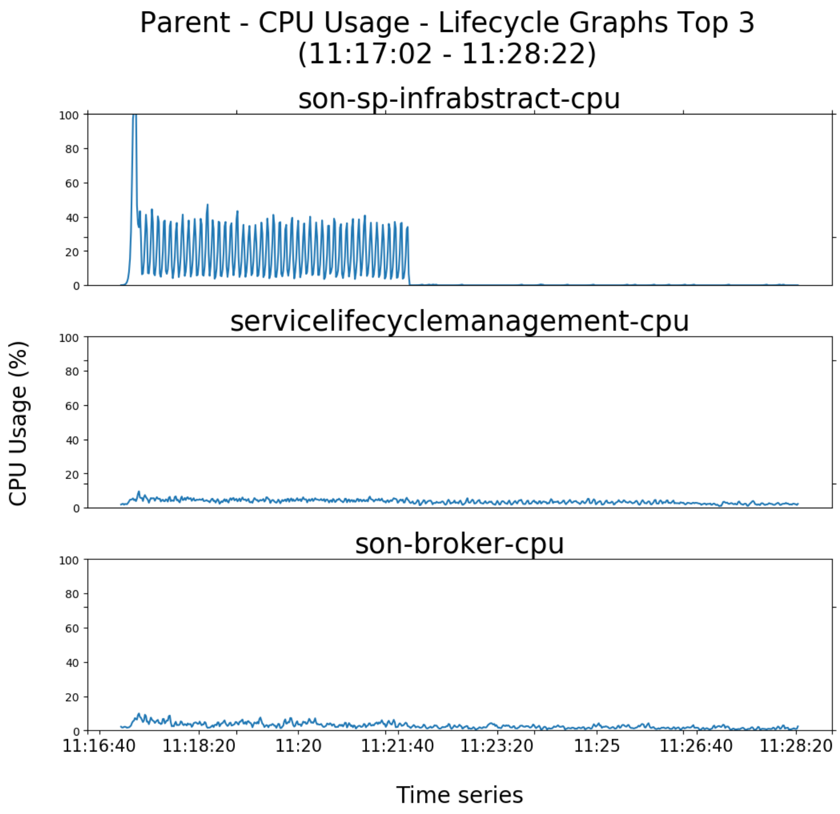

Figure 8.12: Parent Scaling

9
Conclusion

Bibliography

[AHOLA$^+$18] Oscar Adamuz-Hinojosa, Jose Ordonez-Lucena, Pablo Ameigeiras, Juan J Ramos-Munoz, Diego Lopez, and Jesus Folgueira. Automated network service scaling in nfv: Concepts, mechanisms and scaling workflow. *arXiv preprint arXiv:1804.09089*, 2018.

[ALNGT] Mohammad Abu-Lebdeh, Diala Naboulsi, Roch Glitho, and Constant Wette Tchouati. NFV orchestrator placement for geo-distributed systems.

[B$^+$16] Mohamed Boucadair et al. Service function chaining (sfc) control plane components & requirements. *draft-ietf-sfc-control-plane-06 (work in progress)*, 2016.

[DKP$^+$17] Sevil Dräxler, Holger Karl, Manuel Peuster, Hadi Razzaghi Kouchaksaraei, Michael Bredel, Johannes Lessmann, Thomas Soenen, Wouter Tavernier, Sharon Mendel-Brin, and George Xilouris. Sonata: Service programming and orchestration for virtualized software networks. In *Communications Workshops (ICC Workshops), 2017 IEEE International Conference on*, pages 973–978. IEEE, 2017.

[Doca] OSM Documentation. *OSM Schema Documentation.*

[Docb] SONATA Documentation. *SONATA Schema Documentation.*

[dSPR$^+$18] Nathan F Saraiva de Sousa, Danny A Lachos Perez, Raphael V Rosa, Mateus AS Santos, and Christian Esteve Rothenberg. Network service orchestration: A survey. *arXiv preprint arXiv:1803.06596*, 2018.

[Ers13] Mehmet Ersue. Etsi nfv management and orchestration-an overview. In *Proc. of 88th IETF meeting*, 2013.

[ETS14] NFVISG ETSI. Gs nfv-man 001 v1. 1.1 network function virtualisation (nfv); management and orchestration, 2014.

[GMUJ] Jokin Garay, Jon Matias, Juanjo Unzilla, and Eduardo Jacob. Service description in the NFV revolution: Trends, challenges and a way forward. 54(3):68–74.

[HP15] Joel Halpern and Carlos Pignataro. Service function chaining (sfc) architecture. Technical report, 2015.

[LZF$^+$] Guanglei Li, Huachun Zhou, Bohao Feng, Guanwen Li, and Qi Xu. Horizontal-based orchestration for multi-domain SFC in SDN/NFV-enabled satellite/terrestrial networks. 15(5):77–91.

[NFV2] GS NFV. Network functions virtualisation (nfv); architectural framework. *NFV ISG*, 2.

[QE16] Paul Quinn and Uri Elzur. Network service header. *Internet Engineering Task Force, Internet-Draft draft-ietf-sfc-nsh-10*, 2016.

[SON] About SONATA. *SONATA*.

[XWF+15] Wenfeng Xia, Yonggang Wen, Chuan Heng Foh, Dusit Niyato, and Haiyong Xie. A survey on software-defined networking. *IEEE Communications Surveys & Tutorials*, 17(1):27–51, 2015.