COMPARATIVE STUDY OF ORCHESTRATION USING GRPC API AND REST API IN SERVER CREATION TIME: AN OPENSTACK CASE

Hari Krishna S M and Rinki Sharma

Department of Computer Science and Engineering, M. S. Ramaiah University of Applied Sciences, Bengaluru, India

ABSTRACT

Cloud computing is the quick, simple, and economical distribution of computing services. To offer quality services in cloud infrastructure, the NFV Management and Orchestration (MANO) function is a critical task for managing both infrastructure resources and network functions. Since the demand for cloud services is rapidly increasing, cloud service providers are constantly trying to reduce operational costs. Hence the elements of MANO functions have gained critical importance in cloud infrastructure. To use MANO efficiently, there are hypervisors (OpenStack) that act as a middle layer between hardware and software that are used to provide Infrastructure-as-a-Service solutions through a set of interrelated services. These services are managed by an application programming interface (API). The existing orchestration method uses Representational state transfer (REST) API for communicating with core components of a hypervisor. These API have been integral part of creating and deploying a server. To improve the performance, this paper proposes a novel approach for orchestration based on the open-source remote procedure call framework, known as gRPC. To analyse the proposed solution, both REST and gRPC-based orchestration methods are implemented for creating and launching servers using OpenStack cloud computing platform. The server creation time is obtained for different scenarios and SFC use cases. The results show that by using gRPC, the performance in terms of server creation time of an orchestration is improved by up to 27% compared to the traditional REST-based orchestration.

KEYWORDS

Architectural Styles, gRPC, MANO, Network Function Virtualization, Orchestration and REST

1. INTRODUCTION

Through network automation and virtualization, the advantages of Software-Defined Networks (SDN) and Network Function Virtualization (NFV) technologies have significantly aided in the development of open-source software and common network hardware. By offering automatic chaining functions and supporting network connectivity settings, SDN has sped up the adoption of NFV. NFV speeds up the development of new services for service providers while lowering capital and operational costs. A new method for designing, deploying, and managing the network and its services is provided by both SDN and NFV. To integrate the advantages of both technologies, telecommunications providers must manage their network using the NFV Management and Orchestration (MANO) function.

The components of MANO functions have become crucial components in cloud infrastructure because they are a significant resource for controlling infrastructure resources and network functions simultaneously for providing excellent services. Hypervisors serve as a transitional layer between hardware and software in the cloud environment, allowing MANO functions to be used. The most used hypervisor in telecommunications networks is OpenStack. With the help of several connected services, OpenStack offers an Infrastructure-as-a-Service solution. An Application Programming Interface (API) is provided by each service to make this connection easier [1].

1.1. Orchestration Service in Openstack

The orchestration service in OpenStack provides a template to define the cloud application and provides APIs to run and manage the application. Other key OpenStack components are incorporated into a one-file template system. One may create most OpenStack resource types using the templates, including instances, floating IPs, volumes, security groups, etc. The existing orchestration service uses OpenStack-native REST API for communicating with core components of OpenStack.

1.2. REST Based NFV Orchestrator: Characteristics and Challenges

Request-and-response interaction styles are frequently used to describe synchronous communication. One microservice sends a request to another microservice, which must process the response before returning it. It is typical for the re-quester to pause its activity in this style while it waits for the remote server to respond. The most popular framework for synchronous types of communication in microservices is REST API [2].

Each service in a system that uses REST APIs for IPC communication often has a web server operating on a particular port, such 8080 or 443, and each service exposes a set of endpoints to allow interactions with other micro-services and information sharing between them. Through its interface, also referred to as a Web API, the server communicates with the client directly. In orchestration mechanism, different components of a service communicate via REST APIs. Since the demand for cloud services is rapidly increasing, the number of API requests over the network has also increased. This will lead to an increase in server creation time, which indirectly increases the operational cost of the organizations.

1.3. Problem Statements and Objective

Orchestration depends on several components interacting with each other. The existing communication majorly depends on REST API. The REST API has its own limitations when it comes to communication speed and security. Even though REST API runs on HTTP (1.1), and it can be replaced with a better version, the transformation takes lot of time. These problems will affect the performance of the organization with respect to operational costs. The gRPC at its core offers a modern RPC (Remote Procedure Call) framework that simplifies and enhances communication between distributed systems through efficient serialization, protocol optimization, and language-agnostic service definitions. It is well-suited for building microservices, APIs, and other networked applications that require reliable and performant communication between components. To improve the performance by reducing the server creation time, an alternate solution needs to be explored. Recent study shows that in energy cost analysis, gRPC's efficiency and optimized communication patterns might make it a favourable choice for computation offloading in mobile applications compared to protocols like REST [3].

The major objective of this paper to develop and analyse both gRPC API based orchestration and REST API based orchestration methods for creating and launching servers for different scenarios and key applications of SDN and NFV known as Service Function Chaining (SFC).

1.4. Contributions

This paper proposes the following contributions:

1. A novel approach to create a server (instance) using gRPC API on OpenStack.
2. From a performance efficiency standpoint, what are the implications of server creation time in MANO function when utilizing gRPC API for orchestration services in OpenStack cloud computing platform?
3. Comparison analysis of REST and gRPC based orchestration for different use cases of SFC scenarios.

2. BACKGROUND AND RELATED WORK

OpenStack is implemented as a set of services. Depending on the cloud computing platform we are interested in creating, the respective services must be installed. To start basic cloud computing system using OpenStack, need to create a controller node and a compute node. The most crucial node in the OpenStack platform is the controller node. On this node, most of the services are active. Instances of virtual machines are produced on the compute node according to the needs. By configuring OpenStack's Keystone, Glance, Nova, and Neutron services, a controller node can be launched. It is also possible to launch a compute node using the configuration of Neutron and Nova. Virtual server instances can be created quickly and affordably using these minimal services in controller and compute nodes. In OpenStack, there are numerous processes for creating and deleting virtual server instances of various flavors (memory and CPU variants) and images. The OpenStack orchestration procedures are what these processes are known as. A command line instruction or the controller dashboard must be used to launch the creation or deletion of any virtual instances from the controller node.

2.1. Orchestration in Openstack

There are many existing orchestration procedures used by OpenStack, the procedures are grouped based on different categories of creating a service. There are many papers proposed on the comparison of different cloud systems to create an instance. Some focused-on flexibility and composition of service creation in all layers [4]. The paper [5] proposes a framework for power - saving mechanisms using service- level settings. Another paper [6] focused on efficiency by addressing memory and CPU usage. Recently a paper on the comparison of orchestration procedures with respect to delay in service creation time is presented, these methods are well explained using comparison analysis on creating an instance or service. Even though results demonstrate OpenStack-Server method is faster than the rest of the orchestration procedures. This method cannot be used in realistic situations due to the nature of execution [7]. This paper also proposes a better solution for creating a single instance, which is not realistic in the real world. Though the research towards on instance launch time is limited, the paper [8] demonstrates the instance launch time analysis and network errors in OpenStack.

2.2. Orchestration in Service Function Chaining in Openstack

SFC orchestration is one of the key applications in cloud orchestration [9]. The introduction of the End-to-End SFC orchestration framework majorly focused on addressing the NFV challenges, such as ensuring rapid, and reliable NFV deployments at scale. The SFC orchestration framework called XFORCE was focused initially on life cycle management [10]. These frameworks ensured continuous service delivery with higher performance. The paper on OpenSCaaS addresses the concerns about the integration of SDN and NFV by presenting a platform for providing SFC as a service [11]. This led to service providers making use of SDN and NFV functionalities. Another platform was introduced for SFC orchestration across data centres, in which classification is performed at each hop of the SFC [12]. An IETF draft [13] proposes a couple of methods for inter-data centre SFC deployments and another one with a single SFC domain for providing control and visibility to the centralized coordinator for multi-domain SFC. For better utilization of single-domain and multi-domain networks, the research proposes a novel approach to address the joint issues of Virtual Network Function (VNF) assignment and embedding path selection. Another research [14] proposes a heuristic algorithm for redesigning the topology of multiple SFC requests. These strategies contributed to network scalability and reliability.

The work carried out in [15] focuses on VNF orchestration and proposes a dynamic programming-based heuristic algorithm for addressing larger instances of VNF. In [16] orchestration mechanisms are proposed for mitigating congestion in the network. Another approach was proposed to determine good sub -optimal solutions to the VNF's chain placement problem in [17]. Researchers have proposed several ideas to solve the virtual network embedding problem across a multi-domain network [18] [19] [20] [21], however, these ideas cannot be directly used to orchestrate SFC requests. For the efficiency of VNF placement order, the research presented in [22] discusses the domains to share some key information with a third party so that the SFC request to the substrate network can be made globally. Since it is a centralized solution, the privacy of various domains in the network is violated. The concept of distribution for orchestrating VNFs across multiple domains is proposed in [23] [24]. The architectural and optimization models for the automation of SFC, especially in multi-cloud environments are described in [25]. The research concentrates on SFC request topology redesigns in single-domain or multi-domain networks for resource-efficient SFC orchestration.

One of the most recent papers presented systematic literature on SFC allocation of distributed cloud services on an NFV and SDN integrated platform. The authors suggest that a better approach is needed for an efficient solution so SFC management [26]. Another systematic review [27] highlights the problems related to performance, cost, and management of the data centres. Authors in [28] proposed a taxonomy of NFV resource allocation methods, authors also identified problems with towards the efficiency of VNF life cycle management. In this context, a more comprehensive survey on SFC placement and solutions that are focused on operational cost, delay, and revenue generated in distributed scenarios is presented in [29]. The authors in [30] emphasize that the operational cost is one of the most presented metrics along with the acceptance rate, reliability, and execution time.

While the primary studies carried out focused on operational cost, other parameters such as revenue, and its optimization in launching virtual machines to host applications in cloud environments. The paper [31] explores the calculation of total revenue based on serving SFC requests. According to the survey, more than fifty percent of papers focused on operational cost, revenue, and execution time in algorithms. These recent research papers focus on frameworks, topology, VNF placement, and algorithms to reduce operational and expenditure costs. From the study, it is observed that not much research has focused on the APIs used for SFC orchestration.

2.3. Application Programming Interfaces

An API is essential for facilitating communication between various goods and services in NFV environment [32][33]. Most of the research papers on NFV work toward architectural approaches, technology adoptions, functional implementation, and deployment strategies [34]. As the application of SDN and NFV increases, middleware along with the application layer has triggered the interest of the researchers. A comprehensive survey of APIs in SDN and NFV, across different domains, is presented in [35]. The applications presented on papers [36-38] can benefit from the exploration of proposed gRPC-based orchestration in cloud computing platforms.

3. System Design

Existing SFC orchestration in OpenStack is based on REST APIs. Proposing a new orchestration approach based on a gRPC API involves integrating innovative concepts that enhance the efficiency, scalability, and flexibility of orchestrating distributed systems. The gRPC orchestrator leverages gRPC's efficiency and flexibility to enable adaptive versioned API and compatibility orchestration. This research work focuses on estimating the execution time for launching a server based on the user requirement. The performance of SFC orchestration is determined in terms of execution time by using gRPC API for internal and external communication. The general interaction between the client, web framework, SFC engine, SDN, OpenStack, VNF, and Network Function Virtual Infrastructure (NFVI) is presented in Figure 1.

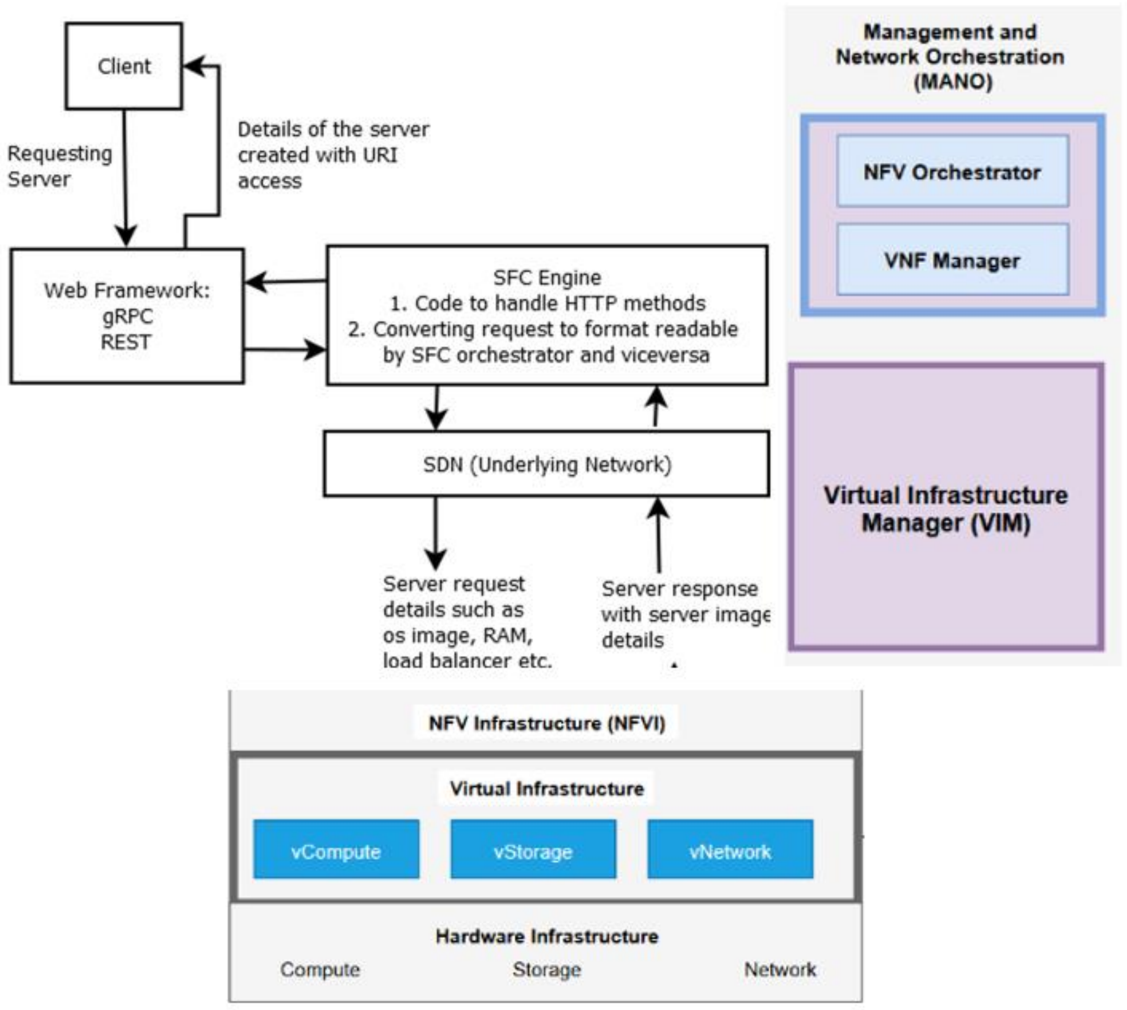

Client
Requesting Server
Details of the server created with URI access
Web Framework: gRPC REST
SFC Engine
1. Code to handle HTTP methods
2. Converting request to format readable by SFC orchestrator and viceversa
SDN (Underlying Network)
Server request details such as os image, RAM, load balancer etc.
Server response with server image details
Management and Network Orchestration (MANO)
NFV Orchestrator
VNF Manager
Virtual Infrastructure Manager (VIM)
NFV Infrastructure (NFVI)
Virtual Infrastructure
vCompute
vStorage
vNetwork
Hardware Infrastructure
Compute
Storage
Network

Figure 1. Orchestration Process in OpenStack

To implement the proposed solution, open-source cloud computing infrastructure software OpenStack is used. SFC orchestration is carried out using OpenStack software. OpenStack contains components such as Glance, Neutron, and Nova that are used for SFC orchestration. The users interact with APIs to interact with SFC components using the SFC orchestrator. The SFC request is sent to the SFC engine (orchestrator) using APIs. The existing well-known frameworks use REST-based APIs. The proposed gRPC-based approach can also be used to do the same orchestration. The components of Nova architecture are described as follows:

- Keystone: This provides identity and authentication for all the OpenStack services.
- Glance: This provides the compute image repository for launching instances such as Ubuntu.
- Neutron: This provides virtual or physical networks. It is responsible for creating network components and management of network topologies.
- Placement: This is used for tracking the inventory of the resources. This component provides overall resource allocation details for a launched instance.

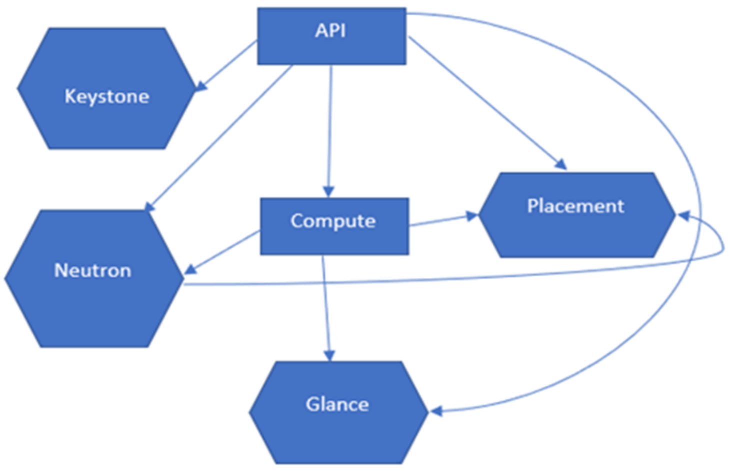


Figure 2. Nova Architecture

As shown in Figure 2, The SFC orchestrator communicates to the components such as Keystone, Glance, and Neutron to get the details for launching an instance. The Nova component provides 'compute' services, which are used to launch instances with all other relevant details from other components.

4. DESIGN AND IMPLEMENTATION

The SFC orchestration communication process is shown in Figure 3, it consists of a client sending a request to the orchestrator.

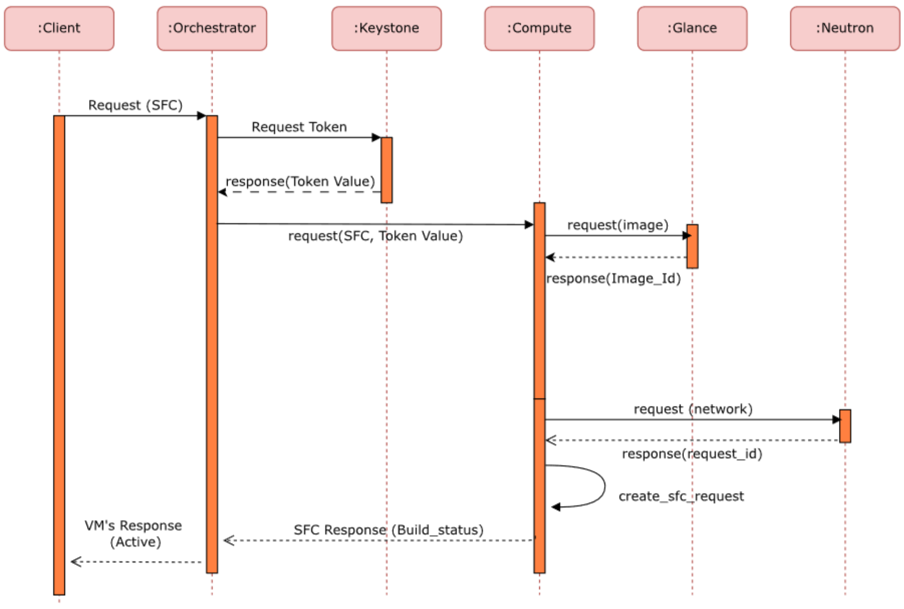
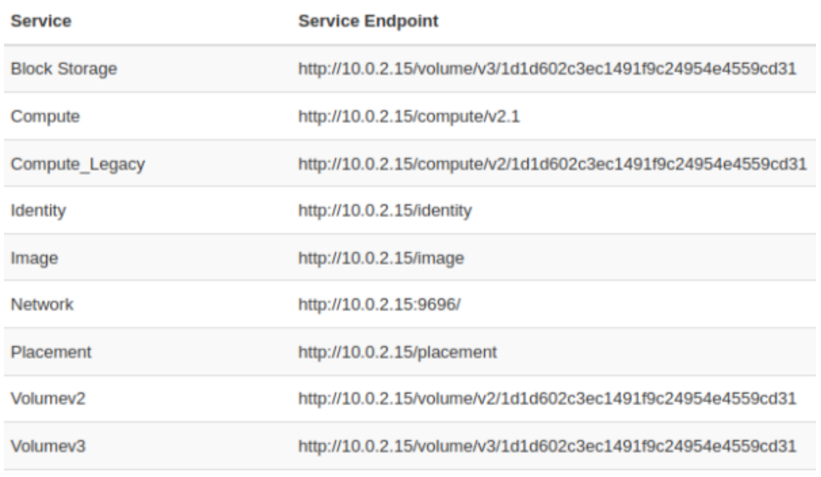


Figure 3. SFC Orchestration Message Flow

The orchestrator communicates to the Keystone component for authentication service and retrieves the token value. Further, the orchestrator sends an SFC request along with the token value to the compute service (Nova component). The compute service fetches the information from the image and network details from the Glance and Neutron components. Once the instance is launched, a response is sent to the client after the status is “ACTIVE”.



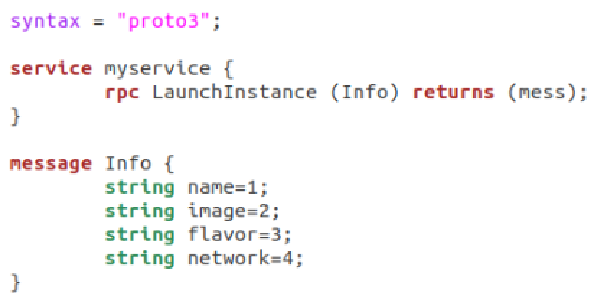
Service	Service Endpoint
Block Storage	http://10.0.2.15/volume/v3/1d1d602c3ec1491f9c24954e4559cd31
Compute	http://10.0.2.15/compute/v2.1
Compute_Legacy	http://10.0.2.15/compute/v2/1d1d602c3ec1491f9c24954e4559cd31
Identity	http://10.0.2.15/identity
Image	http://10.0.2.15/image
Network	http://10.0.2.15:9696/
Placement	http://10.0.2.15/placement
Volumev2	http://10.0.2.15/volume/v2/1d1d602c3ec1491f9c24954e4559cd31
Volumev3	http://10.0.2.15/volume/v3/1d1d602c3ec1491f9c24954e4559cd31

Figure 4. OpenStack Services (REST APIs)

The system is developed on a cloud operating platform called OpenStack. The existing services available in OpenStack are shown in Figure 4. Each service can be accessed by using its service endpoint, known as APIs. These APIs follow traditional REST architectural styles.

4.1. gRPC-based SFC Orchestration

The gRPC and its tools are used to establish the client-server environment. The server and client code are generated from a proto file called service definitions using the protocol buffer compiler in gRPC tools.



```
syntax = "proto3";

service myservice {
        rpc LaunchInstance (Info) returns (mess);
}

message Info {
        string name=1;
        string image=2;
        string flavor=3;
        string network=4;
}
```

Figure 5. Sample proto service definitions

The service definitions file contains the instance name, image id, flavor ID, and network id to provide a structure. The information is defined in a proto file along with the request and response format as shown in Figure 5.

In gRPC, the compiler "protoc" generates code from the proto file shown in Figure 5. It generates the gRPC client and server code with protocol buffer code for populating, serializing, and retrieving the message types. Since clients need to access the gRPC server, the main function needs to be defined in the server file. The server details contain the classes required to be accessed. The server also needs an IP address and port to access.

To access the gRPC server and establish the channel, the gRPC client is initialized with the server address and port number. The request received (instance name, image ID, flavor ID, and network id) for the create method is initialized with the necessary parameters. Once an authentication session is established, requests from the client along with the required parameters are sent to create a method of the server class. Create method creates an instance using the Nova component. Once an instance is launched, the response is sent to the client after the status of the instance is 'ACTIVE'. To enable SFC orchestration using gRPC API in the OpenStack cloud computing platform, the following steps the necessary configuration steps and workarounds for successful integration:

1. Import the necessary libraries for gRPC communication.
2. Define the Protobuf service and message definitions for the resources (networks, subnets, router) as shown in Figure 5.
3. Create a client stub to connect to the gRPC server.
4. Define the authentication payload with the necessary credentials in the Protobuf format.
5. Authenticate and obtain the token by making a gRPC call to the authentication service.
6. Define a gRPC call to create a network:
   a. Create a network request message with the network name and admin state.
   b. Make a gRPC call to create the network using the obtained authentication token.
7. Define similar gRPC calls for creating subnets, a router, and other required resources.
8. Record the start time using the time module.
9. Call the gRPC function to create the network.
10. Retrieve the network IDs using gRPC calls to the network service with the network names.

11. Launch instances using a gRPC call with a request message containing necessary details like name, image reference, flavor reference, and network information.
12. Implement a gRPC call to check the status of launched instances:
    a. Make a gRPC call to obtain server information.
    b. Loop through the servers and check their status until they become active.
13. Call the gRPC function to launch the instances.
14. Call the gRPC function to check the status of the launched instances.
15. Calculate and print the total execution time by recording the end time using the time module and subtracting the start time.

4.2. REST-Based SFC Orchestration

To launch an instance, identity and compute services are used. The identity service (Keystone component) provides the API for client authentication and generates the token value which is used for launching an instance as well as other operations such as getting the image ID (Glance) and network ID (Neutron) from the OpenStack components. The image ID and network ID are essential information needed to launch an instance.

To get the authentication token value, python request module is used to call a POST method via service endpoints (Identity) as shown in Figure 4. The payload contains project details such as project name, ID, and password among others. To launch an instance (server), a payload with the instance name, image ID, flavor ID, and network ID is sent to the compute service endpoint using the request POST method. Once the instance request is sent along with the payload, the server is created, and the response is sent.

5. PERFORMANCE COMPARISON AND SIMULATION RESULTS

To develop and analyze the proposed orchestration model, the system is initially tested on Ubuntu 18 as the platform, employing the Cirros Operating System as the server image with a size of 15.55 MB, allocating 256 MB of Random Access Memory to each server image, ranging from 1 to 4 Virtual CPUs (VCPU), and leveraging the OpenStack Cloud Operating Platform. The performance is obtained on Ubuntu-18 with 8 gigabytes of random-access memory (RAM) and 4 core processors. The results are carried out for the 'N' number of clients and each client contains the 'M' number of instances. Table I shows the simulation range for the 'M' number of instances across the 'N' number of clients. For example: for three clients, the results are obtained for 1, 2, and 3 instances per request each.

Table I. Simulation Range.

Number of Clients (N)	Number of Instances (M)
1	1-5
2	1-4
3	1-3
4	1-2
5	1-2
7	1
10	1

5.1. Experimental Results

The service time to create servers (instances) is recorded for the existing and proposed orchestration model. The results are obtained as per the simulation parameters presented in Table I.

The start-time is recorded when the server receives the request, and the stop-time is recorded when the instance is created, and the status of the instance becomes 'ACTIVE'. The server creation time is tabulated by calculating the difference between the stop time and start time. Table II shows the server creation time for the existing (REST) solution along with the proposed (gRPC) solution. The results show that initially for a single request REST-based SFC orchestration has marginally better execution time than gRPC- based SFC orchestration. But as the number of clients increased, the gRPC-based approach performed better than the REST-based approach.

The results are carried out for the 'N' number of clients and each client contains the 'M' number of instances. Table II shows the simulation range for the 'M' number of instances across the 'N' number of clients. For example: for three clients, the results are obtained for 1, 2, and 3 instances per request each. The average time is calculated for 'M' instances, the results clearly show that gRPC- based orchestration is faster than REST -based orchestration (Ref. Figure 6.)

Table II. Simulation Results.

Number of Instance (M)	**Number of Clients (N)**	**REST-SFC (Seconds)**	**gRPC-SFC (Seconds)**
1	1	6.4	7.6
	3	14.46	15.1
	5	20.5	22.04
	7	34.25	28.23
	10	42.97	38.35
2	1	12.25	10.67
	2	15.9	17.5
	3	21.03	21.56
	4	29.13	28.24
	5	39.66	36.31
3	1	15.91	14.86
	2	27.01	24.16
	3	40.86	33.53
4	1	22.24	18.72
	2	34.8	31.4
5	1	21.96	18.96
	2	39.12	29.1

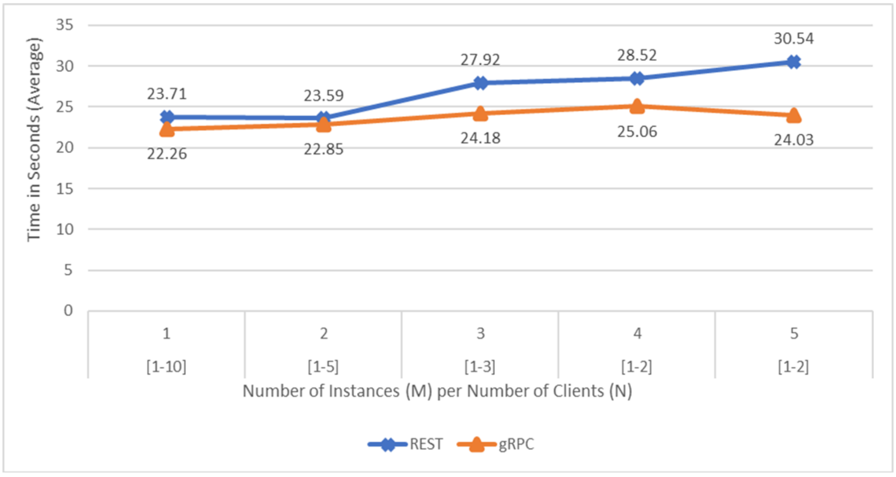


Figure 6. Average execution time for REST and gRPC based orchestration

Based on the initial results, the following observations are obtained:

1. The results show that the server creation time for REST is better for a single client with a minimum instance request. As observed from the obtained results, REST is marginally faster than gRPC for up to five requests. This is due to the characteristic of gRPC, which is not suited for a smaller number of requests. Each request needs to establish a connection with the gRPC server. This initialization time adds up to the delay in establishing the initial requests.

2. The results also indicate that as the number of clients increases, gRPC-based orchestration server creation time gradually reduces when compared to REST -based orchestration. The difference in server creation time is directly proportional to the number of clients requesting a number of instances. This is due to the protocol buffer in gRPC which performs better data serialization, making the payload faster and simpler.

3. Percentage improvement in server creation time is calculated using the equation:

$$(new\ value - old\ value)/old\ value\ * 100\ \% \quad (1)$$

   based on the experimental results, on average the percentage improvement in gRPC based execution time is 11.84% faster than REST based execution time.

5.2. SFC Use Cases Results

Further, the developed solutions were tested on use cases of SFC orchestration. The simple real-world scenarios are designed by considering the common use cases they are received by the cloud operators. Due to the hardware and platform constraints, the use cases are simplified. The designed use case is follows:

Use case 1. The user request servers on multiple networks that are connected. This scenario is most used in simpler applications for a limited amount of time such as running some experiments or training the model on the cloud. Figure 7 shows the topology for multiple servers created at multiple networks (private, private2 and shared) that are connected through router (r1).

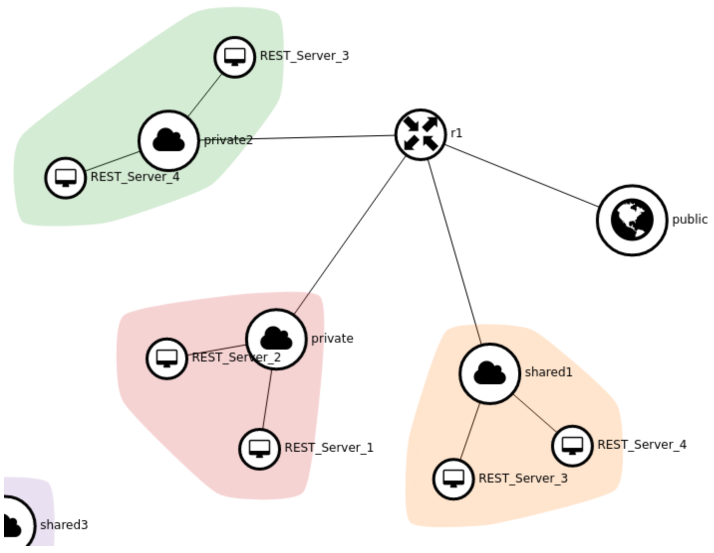


Figure 7. Use case 1 Topology

Use case 2. This scenario is like use case 1 except it contains one of the most important and used features known as load balancer. The user requests a load balancer as a service since it can handle high-level business where the number of clients accessing the server at the same time is high in demand. These services involve most of the business. Figure 8 shows multiple users requesting multiple servers along with a load balancer.

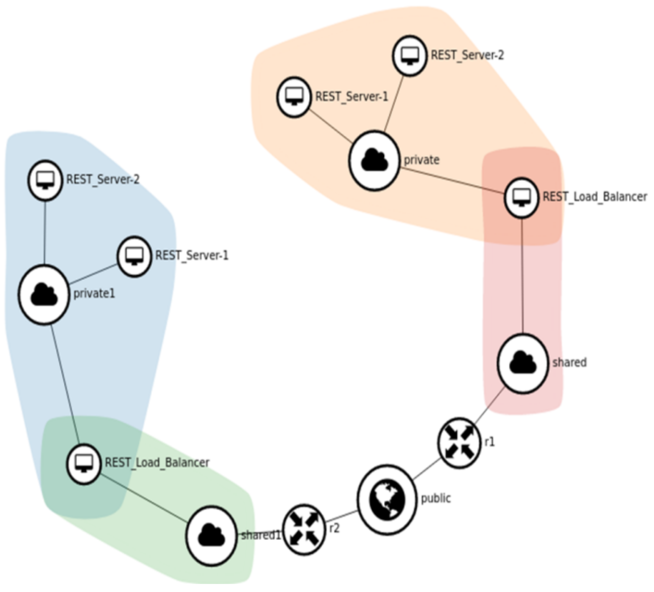


Figure 8. Use case 2 Topology

Use case 3. This scenario is built on top of the use case 2 topology. The user requests multiple servers along with a load balancer and Domain Name System (DNS) as a service. These services are commonly used in deploying hosts on a private a DNS server. Figure 9 depicts the topology of a client requesting services with two servers connected from one network to DNS server (private to shared1) and followed by a load balancer on another network (shared).

Use case 4. Similar to use case number 3, the user requests multiple servers with DNS, load balancer, and along with firewall as a service. Most of the clients prefer their own firewall service. Figure 10 shows the sample topology where users request such services.

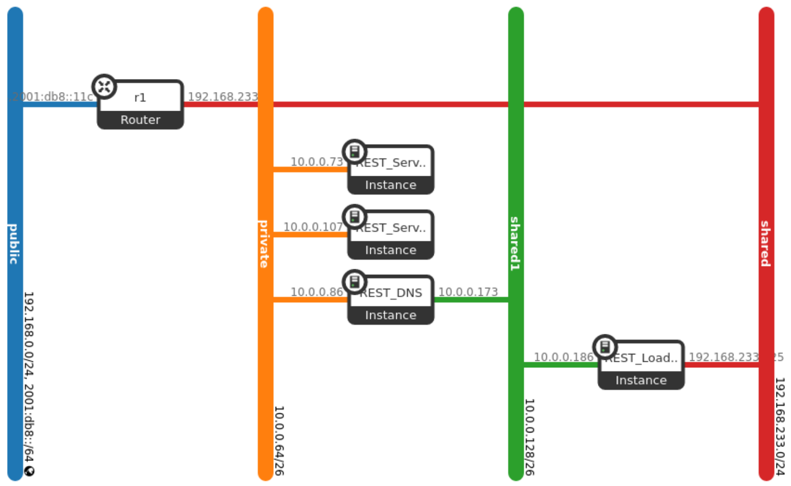


Figure 9. Use case 3 Topology

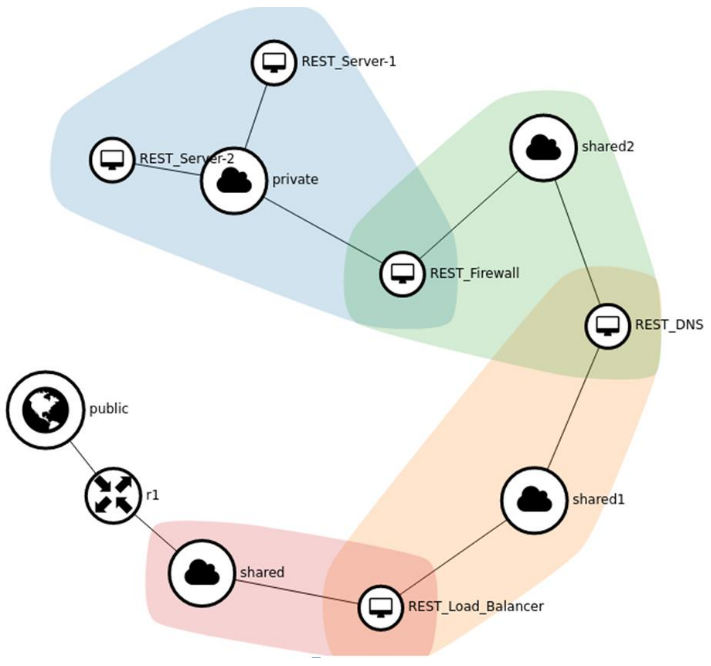


Figure 10. Use case 4 Topology

The use cases shown in from Figure 7 to Figure 10 are designed and implemented in REST and gRPC -based SFC orchestration cloud platforms using OpenStack. The service time for creating and launching a service is calculated on both platforms. The results are tabulated in Table III.

Table III. Use Case Simulation Results.

Use case No.	Number of Clients (N)	REST-SFC (Seconds)	gRPC-SFC (Seconds)	Difference (Seconds)
1	1	8.56	6.63	1.93
	2	13	9.83	3.17
2	1	10.74	8.84	1.9
	2	18.72	13.79	4.93
3	1	13.1	9.44	3.66
	2	22.44	14.6	7.84
4	1	17.51	13.69	3.82
	2	25.59	17.32	8.27

Table III shows the results of each use case conserving server creation time. Initially, results are obtained for a single request from a single client. Then the number of clients is increased from one to two. Figure 11 and Figure 12 show the results of service execution time for developed use cases. The results show that gRPC-based orchestration has better results than REST-based orchestration.

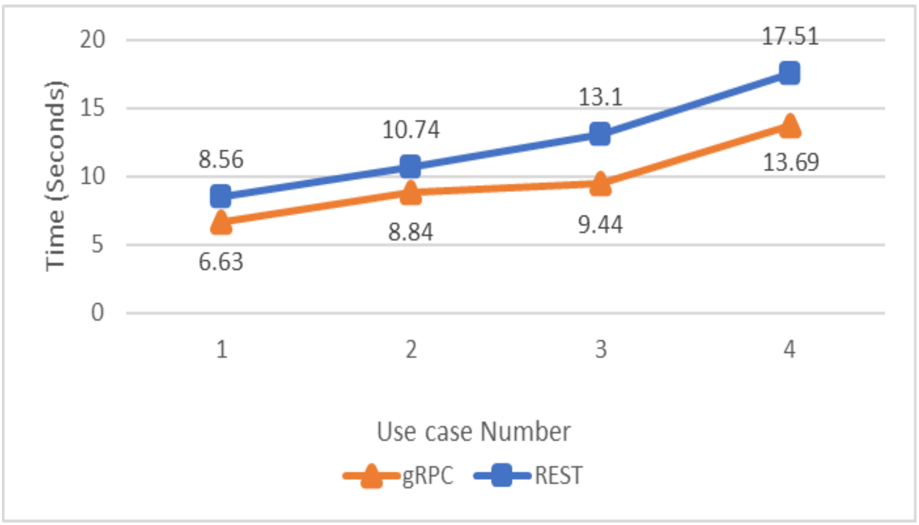


Figure 11. gRPC vs REST-based service execution time for designed use cases (Single Request)

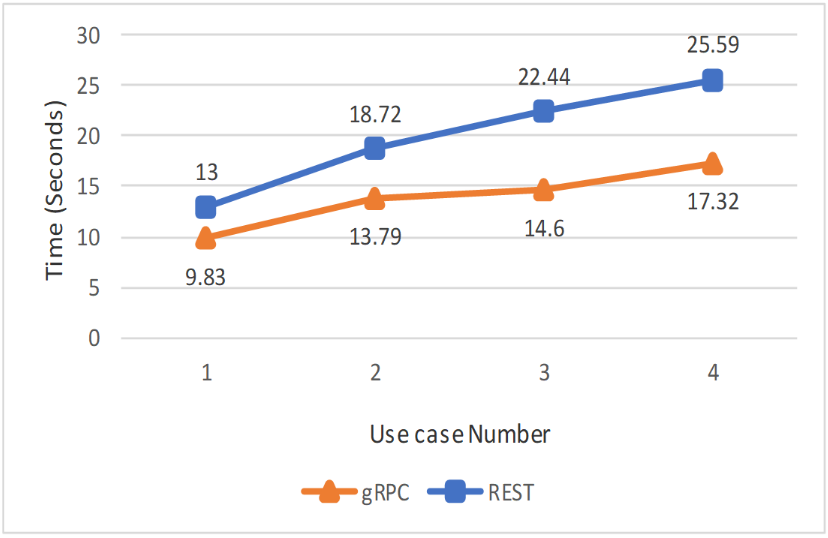


Figure 12. gRPC vs REST-based service execution time for designed use cases (Two Request)

The gRPC performance is better as the number of clients increases. The results in Table III show the difference in seconds for a single request and two requests. As the data shows the difference is increasing for a higher number of clients (see Figure 13).

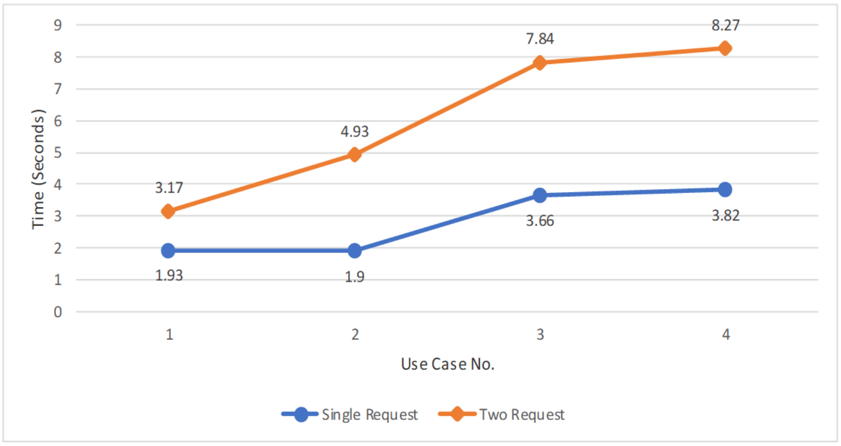


Figure 13. gRPC Performance

Figure 13 shows the gRPC performance over REST in terms of service execution time. The use case results data shows that for two requests, the gRPC results in are improved (at least twice) over a single request.

Based on the use case results analysis (ref. Table III), the following observations are obtained:

1. The execution time for based orchestration is better than REST -based orchestration.
2. The performance of gRPC is improved with use cases containing multiple parameters (such as servers, load balancers, DNS, and firewall). The time differences are increasing as the number of parameters in use cases increases.
3. The performance of gRPC is also improved when the same set of requirements are requested by multiple clients. The time differences were significantly high when the number of clients was increased.

4. Percentage improvement in server creation time is calculated using the equation (1). Based on the use cases simulation results, on average the percentage improvement over gRPC-based execution time is 27.37% faster than REST- based execution time.

The improvement in server creation time is due to gRPC's efficient communication and coordination between services, gRPC's performance-oriented features contribute to quicker execution times by minimizing data transfer overhead, reducing latency, and enabling effective concurrent communication patterns.

5. CONCLUSION

Orchestration is an essential part of any cloud operating environment. Especially cloud platforms with high demand for serving multiple clients. The server creation time plays a crucial role in cloud-based services, such as service function chaining. Thus, a reduction in server creation time plays is a very important feature. To reduce the execution time, a novel gRPC- based orchestration method was developed and compared to the existing REST-based method. REST and gRPC-based SFC orchestration are analyzed in terms of server creation time on the OpenStack cloud operating platform. The Following observation are made from the obtained results:

1. The experimental results show that for a single request, the execution time for REST-based SFC orchestration is better compared to that of gRPC. This difference is marginal, and it is insignificant as in reality, such use cases do not exist in common scenarios.
2. The initial simulation results show gRPC was faster, when:
    - The number of clients has increased.
    - The number of requirements (servers) has increased from a single client.
    - The server creation time of gRPC-based orchestration is 11% faster than REST-based orchestration.
3. The SFC use case simulation results show that gRPC results were much clearer and better understanding, when:
    - The number of requirements (servers, DNS, firewall, load balancer) is increased from a single client.
    - The number of clients requesting the same requirements has increased.
    - The server creation time of gRPC-based SFC orchestration is 27% faster than REST-based SFC orchestration.

It is seen that the measures taken to reduce server creation time have proved to be effective. A comparison with traditional REST-based orchestration method shows that the gRPC-based orchestration method exhibits a reduction of almost up to 27% in the server creation time that is required to create instance and as well as to ensure an instance launch is successful. This corresponds to a reduction in total overall execution time.

CONFLICTS OF INTEREST

The authors declare no conflict of interest.

REFERENCES

[1] Kaur K, Mangat V and Kumar K 2022 A review on Virtualized Infrastructure Managers with management and orchestration features in NFV architecture. Computer Networks, p.109281.

[2] Richardson C 2018 Microservices patterns: with examples in Java. Simon and Schuster

[3] Chamas C L, Cordeiro D and Eler M M 2017 Comparing REST, SOAP, Socket and gRPC in computation offloading of mobile applications: An energy cost analysis. IEEE. Latin-American Conference on Communications (LATINCOM) (pp. 1-6).

[4] Di Martino B, Cretella G and Esposito A 2015 Defining cloud services workflow: a comparison between TOSCA and OpenStack Hot. IEEE. International Conference on Complex, Intelligent, and Software Intensive Systems (pp. 541-546).

[5] Carrega A, Repetto M, Robino G and Portomauro G 2018 OpenStack extensions for QoS and energy efficiency in edge computing. IEEE. International Conference on Fog and Mobile Edge Computing (FMEC) (pp. 50-57).

[6] Carella G, Corici M, Crosta P, Comi P, Bohnert T M, Corici A A et al 2014 Cloudified IP multimedia subsystem (IMS) for network function virtualization (NFV)-based architectures. IEEE. Symposium on Computers and Communications (ISCC) (pp. 1-6).

[7] Mandal P and Jain R 2019 Comparison of OpenStack orchestration procedures. IEEE. International Conference on Smart Applications, Communications and Networking (SmartNets) (pp. 1-4).

[8] Ahmed J, Malik A, Ilyas M U and Alowibdi J S 2019 Instance launch-time analysis of OpenStack virtualization technologies with control plane network errors. Computing, 101(8), pp.989-1014.

[9] Bhamare D, Jain R, Samaka M and Erbad A 2016 A survey on service function chaining. Journal of Network and Computer Applications, 75, pp.138-155.

[10] Medhat A M, Carella G A, Pauls M and Magedanz T 2017. Extensible framework for elastic orchestration of service function chains in 5g networks. IEEE. Network function virtualization and software defined networks (NFV-SDN) (pp. 327-333).

[11] Ding W, Qi W, Wang J and Chen B 2015 OpenSCaaS: an open service chain as a service platform toward the integration of SDN and NFV. IEEE Network, 29(3), pp.30-35.

[12] Ren J, Zhang Y, Zhang K and Shen X 2015 Exploiting mobile crowdsourcing for pervasive cloud services: challenges and solutions. IEEE Communications Magazine, 53(3), pp.98-105.

[13] Ayoubi S, Sebbah S and Assi C 2017 A logic-based benders decomposition approach for the VNF assignment problem. IEEE Transactions on Cloud Computing, 7(4), pp.894-906.

[14] Ye Z, Cao X, Wang J, Yu H and Qiao C 2016 Joint topology design and mapping of service function chains for efficient, scalable, and reliable network functions virtualization. IEEE Network, 30(3), pp.81-87.

[15] Bari M F, Chowdhury S R, Ahmed R and Boutaba R 2015 On orchestrating virtual network functions. IEEE. International Conference on Network and Service Management (CNSM) (pp. 50-56).

[16] Elias J, Martignon F, Paris S and Wang J 2015 Efficient orchestration mechanisms for congestion mitigation in NFV: Models and algorithms. IEEE Transactions on Services Computing, 10(4), pp.534-546.

[17] Khebbache S, Hadji M and Zeghlache D 2017 Virtualized network functions chaining and routing algorithms. Computer Networks, 114, pp.95110.

[18] Alaluna M, Ferrolho L, Figueira J R, Neves N and Ramos F M 2020 Secure multi-cloud virtual network embedding. Computer Communications, 155, pp.252-265.

[19] Mano T, Inoue T, Ikarashi D, Hamada K, Mizutani K and Akashi O 2016 Efficient virtual network optimization across multiple domains without revealing private information. IEEE Transactions on Network and Service Management, 13(3), pp.477-488.

[20] Dietrich D, Rizk A and Papadimitriou P 2015 Multi-provider virtual network embedding with limited information disclosure. IEEE Transactions on Network and Service Management, 12(2), pp.188-201.

[21] Wang Y, Lu P, Lu W and Zhu Z 2017 Cost-efficient virtual network function graph (vNFG) provisioning in multidomain elastic optical networks. Journal of Lightwave Technology, 35(13), pp.2712-2723.

[22] Dietrich D, Abujoda A, Rizk A and Papadimitriou P 2017 Multiprovider service chain embedding with nestor. IEEE Transactions on Network and Service Management, 14(1), pp.91-105.

[23] Abujoda A and Papadimitriou P 2016 DistNSE: Distributed network service embedding across multiple providers. IEEE. International Conference on Communication Systems and Networks (COMSNETS) (pp. 1-8).

[24] Zhang Q, Wang X, Kim I, Palacharla P and Ikeuchi T 2016 Vertex-centric computation of service function chains in multi-domain networks. IEEE. Netsoft conference and workshops (NetSoft) (pp. 211-218).

[25] Bhamare D, Jain R, Samaka M, Erbad A 2016 A survey on service function chaining. Journal of Network and Computer Applications 75, 138–155

[26] Bonfim M S, Dias K L, Fernandes S F 2019 Integrated nfv/sdn architectures: A systematic literature review. ACM Computing Surveys (CSUR) 51(6), 1–39

[27] Souza R, Dias K, Fernandes S 2020 Nfv data centers: A systematic review. IEEE Access, 51713–51735

[28] Schardong F, Nunes I, Schaeffer-Filho A 2020 Nfv resource allocation: A systematic review and taxonomy of vnf forwarding graph embedding. Computer Networks p. 107726

[29] Santos G L, Bezerra D D F, Rocha E D S, Ferreira L, Moreira A L C, Gon et al 2022 Service Function Chain Placement in Distributed Scenarios: A Systematic Review. Journal of Network and Systems Management, 30(1), pp.1-39.

[30] Badshah A, Ghani A, Shamshirband S, Chronopoulos A T 2019 Optimising infrastructure as a service provider revenue through customer satisfaction and efficient resource provisioning in cloud computing. IET Communications. 13(18), 2913–2922

[31] Xie Y, Wang S, Dai Y 2020 Revenue-maximizing virtualized network function chain placement in dynamic environment. Future Generation Computer Systems

[32] What is an API? (Application Programming Interface) MuleSoft 2021

[33] Ananth M D and Sharma R 2017 Cost and performance analysis of network function virtualization-based cloud systems. IEEE. International Advance Computing Conference (IACC) (pp. 7074).

[34] Nguyen V G, Brunstrom A, Grinnemo K J and Taheri J 2017 SDN NFV-based mobile packet core network architectures: A survey. IEEE Communications Surveys Tutorials, 19(3), pp.1567-1602.

[35] Krishna S H and Sharma R 2021 Survey on application programming interfaces in software defined networks and network function virtualization. Global Transitions Proceedings, 2(2), pp.199-204.

[36] Hernandez, J.F., Lorios, V.M., Avalos, M. and Silva-Lepe, I., 2016. Infrastructure of services for a smart city using cloud environment. International Journal of Computer Networks & Communications (IJCNC), 8(1), pp.105-119.

[37] Mulla, B.P., Krishna, C.R. and Tickoo, R.K., 2020. Load balancing algorithm for efficient VM allocation in heterogeneous cloud. International Journal of Computer Networks & Communications (IJCNC) Vol, 12.

[38] Le, H.N. and Tran, H.C., 2022. Ita: The Improved Throttled Algorithm of Load Balancing On Cloud Computing. International Journal of Computer Networks & Communications (IJCNC) Vol, 14.

AUTHORS

Mr. Hari Krishna S M is presently working as Assistant Professor in the Department of Computer Science and Engineering, M S Ramaiah University of Applied Sciences, Bangalore. I am currently pursuing my Ph.D. from the same university. Currently my area of research is Software Defined Networks, Network Function Virtualization, Service Function Chaining and Architectural styles.

Dr. Rinki Sharma is presently working as Professor and Head of Computer Science and Engineering department, Ramaiah University of Applied Sciences, Bangalore. She obtained her Ph. D from Coventry University, United Kingdom in the year 2015. She has over 16 years of work experience in industry, teaching, and research. Apart from teaching and research she has held other vital positions at the Ramaiah University of Applied Sciences, Bangalore. She has been granted 5 international patents, has published 5 book chapters, and 15 papers in reputed conferences and journals.