A DNS Based Architecture for Publication and Discovery of Virtual Network Functions in Content Delivery Networks

Sandhya Shanmugasundaram

A Thesis

in

The Department

of

Electrical & Computer Engineering

Presented in Partial Fulfillment of the Requirements for the

Degree of Master of Applied Science at

Concordia University

Montréal, Québec, Canada

October 2017

CONCORDIA UNIVERSITY
SCHOOL OF GRADUATE STUDIES

This is to certify that the thesis prepared

By: **Sandhya Shanmugasundaram**

Entitled: "**A DNS Based Architecture for Publication and Discovery of Virtual Network Functions in Content Delivery Networks**" and submitted in partial fulfillment of the requirements for the degree of

Master of Applied Science

Complies with the regulations of the University and meets the accepted standards with respect to originality and quality.

Signed by the final examining committee:

_____ Chair
Dr. R. Raut

_____ Internal Examiner
Dr. A. Agarwal

_____ External Examiner
Dr. J. Rilling

_____ Supervisor
Dr. R. Glitho

Approved by: _____

Chair of Department or Graduate Program Director

_____2017 _____
Dean of Faculty

ABSTRACT

A DNS Based Architecture for Publication and Discovery of Virtual Network Functions in Content Delivery Networks

Sandhya Shanmugasundaram

In recent years, Network Function Virtualization (NFV) has become very popular for many reasons including flexibility and cost efficiency. Virtual Network Function (VNF) is the implementation of network functions using software which is decoupled from the underlying hardware. Content Delivery Network serves content to end-users with high availability and high performance. In addition to content, they also offer value-added services (video overlay, compressed video, video transcoding, etc.) to end-users. Publication and discovery mechanism is very much substantial to provide meaningful services to the end-users. Incorporating publication and discovery mechanisms in NFV can play a crucial role in CDN to provide on-demand value-added services. However, publication and discovery mechanisms for VNF remains a big challenge. As the existing researchers do not consider an exhaustive way of describing Virtual Network Functions (VNFs), VNF descriptors include details concerning VNF deployment but fail to include functional and non-functional specifications of VNF, which increases an inefficient utilization of VNFs. Another challenge is that the current discovery requires VNF consumers to select the VNF manually rather than having an automated mechanism to discover VNF. Furthermore, publication of VNF and its details are proprietary, which brings difficulty for VNF consumers in discovering and utilizing

them. This thesis aims to propose a solution for efficient description, publication, and discovery of VNFs.

The main contribution of this thesis is twofold. First, we have proposed a VNF descriptor model including its properties and operations for efficient utilization. Second, we have proposed an architecture for publication of VNF descriptors and for discovering them. The proposed architecture use Domain Name System as a foundation. We have built a proof-of-concept prototype for value added service provisioning of VNF using the BIND open source server. In order to evaluate the viability of the proposed architecture, we have made performance measurement of the implemented prototype.

Acknowledgments

Firstly, I would like to express my sincere gratitude to my advisor Dr. Roch Glitho for the continuous support of my Masters study and research, for his patience, encouragement, and immense knowledge. His guidance and constructive criticisms helped me in all the time of research and writing of this thesis. I could not have imagined having a better advisor and mentor for my Masters study.

Besides my advisor, I would also like to thank Dr. Sami Yangui, who provided valuable insights and expertise that greatly assisted the research. I thank my fellow lab mates in Telecommunication Service Engineering (TSE) Laboratory for their continuous motivation and questions that stimulated discussions and incented me to widen my research from various perspectives. I am grateful to Dr. Rilling and Dr. Agarwal for serving as members of my thesis committee.

Last but not least, I must express my very profound gratitude to my parents Shanmugasundaram and Anusya and to my fiancé Dinesh Damodaran for providing me with unfailing support and continuous reassurance throughout my years of study and through the process of researching and writing this thesis. This accomplishment would not have been possible without them. Thank you.

Table of Contents

List of Figures

List of Tables

Acronyms and abbreviations

AVI	Audio Video Interleave
API	Application Program Interface
BIND	Berkeley Internet Name Domain
CAPEX	Capital Expenditure
CDN	Content Delivery Network
CLI	Command Line Interface
DNS	Domain Name System
DOM	Document Object Model
ETSI	European Telecommunications Standards Institute
FTP	File Transfer Protocol
GB	Giga Byte
GUI	Graphical User Interface
HOT	Heat Orchestration Template
HTTP	Hyper Text Transfer Protocol
IDE	Integrated Development Environment
IEEE	institute of Electrical and Electronics Engineers
IP	Internet Protocol

JSDT	JavaScript Development Tools
JSON	JavaScript Object Notation
MANO	Management And Orchestration
NAT	Network Address Translation
NFV	Network Function Virtualization
NFVI	Network Function Virtualization Infrastructure
NS	Name Server
NSD	Network service Descriptor
OPEX	Operational Expenditure
OS	Operating System
PTR	Pointer Record
REST	Representational State Transfer
RPC	Remote Procedure Calls
SAVI	Smart Applications on Virtual Network
SFTP	Secure File Transfer Protocol
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
TOSCA	Topology and Orchestration Specification for Cloud Applications

UDDI	Universal Description, Discovery, and Integration
URI	Uniform Resource Identifier
VM	Virtual Machine
VNF	Virtual Network Function
VNFD	Virtual Network Function Descriptor
W3C	World Wide Web Consortium
WADL	Web Application Description Language
WSDL	Web Service Definition Language
XML	Extensible Markup Language
XSD	XML Schema Definition
YAML	Ain't Markup Language

Chapter 1

1. Introduction

In this chapter, we provide an overview of the key concepts related to our research domain such as Content Delivery Network, Network Function Virtualization and Domain Name System in the first section. The second section presents the motivation and the problem statement. A summary of thesis contributions is presented in the third section. Finally, we conclude this chapter by introducing the thesis organization.

1.1 Definitions

We provide definition to the four key concepts that are related to our research.

1.1.1 Content Delivery Network

CDN is a network of servers designed to deliver content such as static web pages, images, audio, video and streaming media to end-users [1]. CDN also offers various value added services such as advertisement overlay, transcoding video formats, video compressing, etc. CDN helps in reduced latency and high availability by distributing the content close to end-users.

1.1.2 Network Function Virtualization

NFV is a technology which allows softwarization of network functions running on commodity devices such as servers, switches, and storage by leveraging standard IT virtualization and cloud technologies [2]. Virtual Network Functions (VNFs) are software which runs on virtual machines

to handle network functions on top of the hardware infrastructure [3]. Since NFV virtualize network functions and offer them as software, it provides advantages such as reduced deployment time, flexible service, and reduced CAPEX and OPEX costs for the operator.

1.1.3 Domain Name System

DNS is an Internet service which helps in resolving hostnames to IP address. DNS is a globally distributed, scalable, reliable, dynamic database consisting of domain name space, servers and resolvers [4]. DNS has evolved from simple name to IP translation, to enable sophisticated "look up" applications such as voice, data, multimedia, and security [5].

1.2 Motivation and Problem Statement

Virtual Network Functions (VNF) are gaining popularity and used for provisioning services such as firewall, transcoder, mixer, Network Address Translation (NAT) devices, etc. VNFs are described with manifests and published in dedicated repositories by developers. These VNFs can be offered to VNF consumers. Currently, VNF consumers can select VNF among several published VNFs without considering their requirements. Selected VNFs are further downloaded in a target environment and executed to offer service. There are several challenges associated with current mechanism. One of them is the lack of rigorous descriptions for VNFs. Current description models describe VNFs more specific to deployment and fail to include functional and non-functional specifications. Lack of enough information eventually makes it difficult in choosing the appropriate VNF. Another challenge is the vendor lock-in of VNF repositories. VNF repositories are proprietary, making it time-consuming for end-users to find and access the required VNF since they have to evaluate descriptors available in each repository. In addition, vendor lock-in feature

also prevents the repository federation and cooperation among developers in publishing VNF descriptors.

In order to address these challenges, we need an exhaustive descriptor model to assist in the precise discovery of VNF. Also, we need a sound architecture that enables efficient publication of VNFs and discovery procedures associated with utilizing the most relevant VNF based on user's requirement.

1.3 Thesis Contributions

The contributions of this thesis are as stated below:

- A set of requirements for the publication and discovery of VNFs.
- A review of the state-of-the-art with an evaluation summary based on defined requirements
- A description model of VNFs uniquely for the purpose of publication and discovery.
- An architecture for publication and discovery of VNFs.
- A prototype of the proposed architecture evaluated using different performance metrics.

1.4 Thesis Organization

The rest of the thesis is organized as follows:

Chapter 2 discusses the key concepts related to our research domain in more details.

Chapter 3 introduces the scenarios and the set of requirements derived from the scenarios. The state-of-the-art solutions are reviewed and evaluated against the requirements.

Chapter 4 presents the proposed description model for VNFs used in publication and discovery.

Chapter 5 presents the proposed architecture for publication and discovery of VNFs. Architectural components, as well as the proposed interfaces, are discussed.

Chapter 6 describes the implementation architecture and technologies used for the proof-of-concept prototype. This chapter also includes performance measurement of the implemented prototype.

Chapter 7 concludes the thesis by giving a summary of the overall contributions and identifies future research directions.

Chapter 2

2. Background

This chapter presents the background concepts relevant to research domain of this thesis. The following concepts are explained: content delivery networks, network function virtualization, and domain name system. In the end, we summarize the chapter

2.1 Content Delivery Networks

In this section, we provide an overview of Content Delivery Networks. We first briefly introduce the concept of CDN, then introduce its components, functionality and different types of CDN. We finally discuss the advantages of CDN.

2.1.1 A Brief Introduction to CDN

Content Delivery Network (CDN) is a globally distributed network made up of proxy servers over the internet that aims to provide seamless and efficient content delivery to end users with high performance and availability [6]. Content can be broadly classified as encoded media and metadata [7]. Encoded media includes static, dynamic, and continuous media data (e.g. audio, video, documents, images and Web pages).Metadata is the content description that allows identification, discovery, and management of multimedia data. Content can be either pre-recorded or can be retrieved from live sources.

There are three main business actors in a CDN system. They are: Content provider, CDN provider and end-user. Content Provider is one who delegates the URI namespace of the content to be

distributed. CDN Provider is considered to be an organization who provides infrastructure facilities to the content provider. End User access content from the content provider's website.

2.1.2 Architectural components of CDN

A typical CDN consists of the following CDN components (depicted in Figure 2-1): Content delivery component, Distribution component, Request routing component and Accounting component [8].

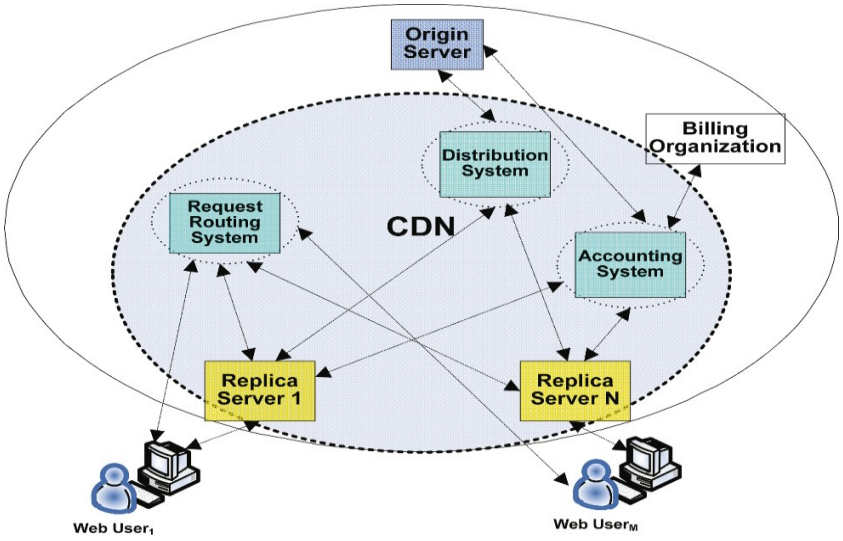

Figure 2-1: Architectural components of CDN [8]

- Content-delivery component consists of the origin infrastructure (Web servers, application servers, and databases hosted by the content provider) and edge infrastructure (a set of replica servers hosted by the CDN provider). The origin server delegates the URI namespace of the content to the request-routing component and publish the content into the distribution component

- Distribution component distributes content from the origin server to the replica servers and ensures consistency of the content in the caches. It also provides feedback to the

request-routing component to assist replica server selection for delivering content to end-users.

- Request-routing component receives end-user requests. They redirect the request to appropriate replica servers. It interacts with the distribution component to keep the content up-to-date on the replica servers. The replica servers then deliver the requested content to end-users and send accounting information to the accounting component.

- Accounting component maintains logs of end users and maintains statistics about usage for CDN resources. These data are used for traffic reporting and for billing purposes by the origin server or a third-party organization. These statistics are also used as feedback to the request-routing component.

2.1.3 Functionality of CDN

The important functionalities of CDN are as follows [8]:

- **Request redirection and content delivery** by forwarding requests to the closely located surrogate server to overcome congestion.
- **Content outsourcing and distribution** by caching and/or replicating content to distributed surrogate servers.
- **Content negotiation** by offering services to meet the needs of an end user.
- **Management** by managing the network components, to handle accounting, and to monitor and report on content usage.

2.1.4 Different types of CDN

CDNs are broadly categorized as Commercial CDN, Academic CDN, and Cloud-Based CDN. A brief explanation of them is as follows.

- **Commercial CDN:** Most of the operational CDNs are developed by commercial companies [9]. These are stable in nature. Examples include Akamai, Limelight Networks, Mirror Image, EdgeStream.
- **Academic CDN:** Academic CDNs are developed as a result of research initiatives. They mainly use P2P technologies. They follow a decentralized approach and request load is spread across all the participating hosts, and the system can handle node failures and load surges. They are effective for static content only, and cannot handle dynamically generated content due to the lack of caching nature. Examples include Globule [10], Coral [11], CoDeeN [12], COMODIN [13], etc.
- **Cloud-Based CDN**: Cloud Based CDN's [14] inherit traditional CDNs model by exploiting clouds for content and Web application delivery, as well as storage, raw computing, or access to any number of specialized services and uses pay-as-you-go pricing model. This improves cost efficiency, scalability, and quick delivery. Examples include MetaCDN, Amazon, Voxel, etc.

2.1.5 Advantages of CDN

CDN offers several important advantages [15]. They are:

- **Reduces operational cost** by decreasing the customer need in website infrastructure

- **Decreases congestion** since the content is distributed closer to the end user and eventually reducing load on origin server
- **Improves content delivery quality, speed, and reliability**
- **Minimize load** on the origin server.

2.2 Network Function Virtualization

In this section, we present a general overview of Network Function Virtualization (NFV) along with architectural components, vendor-specific implementations, use cases and its advantages.

2.2.1 A Brief introduction to NFV

Network Function Virtualization was introduced by European Telecommunications Standards Institute (ETSI) [16]. NFV [17] is a network architecture which aims in virtualizing network services by decoupling software applications from the underlying hardware. NFV proposes that all functions performed by network nodes should be defined in a virtualizable fashion and the functional blocks are termed as Virtual Network Function (VNF) [18]. Through NFV, virtual network functions can be instantiated and chained in various locations such as data centers, network nodes, and end-user premises. Orchestrator is responsible for managing the lifecycle of VNFs. Figure 2-2 describes the concept of NFV and Figure 2-3 represents the NFV domains respectively.

NFV defines Virtual Network Function Descriptors (VNFD) which describe on deploying a single VNF. Network Service Descriptors (NSD) templates define a set of VNFDs. NSD templates are used to describe and design network services by providing rules for interconnection of VNFs.

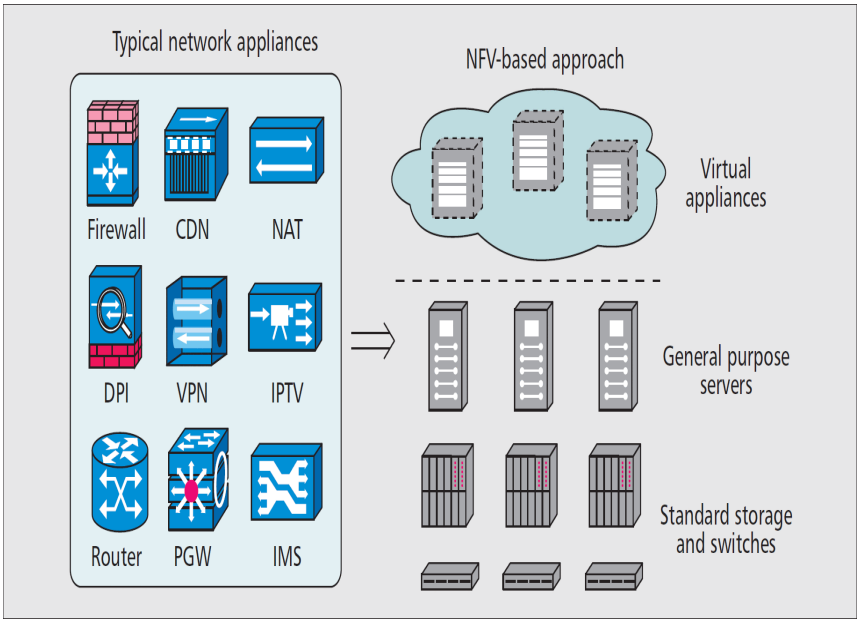

Figure 2-2: Network Function Virtualization [31]

Broadly, NFV includes the following architectural components [19]:

1. A software implementation of a network function (Virtualized Network Function) that is able to run over an NFV infrastructure.

2. The NFV Infrastructure (NFVI), consisting of the hardware resources where virtual networks can reside on and run. NFVI aids in the execution of VNFs.

3. An NFV Management and Orchestration module (MANO) is assigned to manage and orchestrate the life-cycle of virtual network functions, life-cycle management of physical and/or software resources which support infrastructure virtualization. MANO will also be responsible for management tasks in NFV framework.

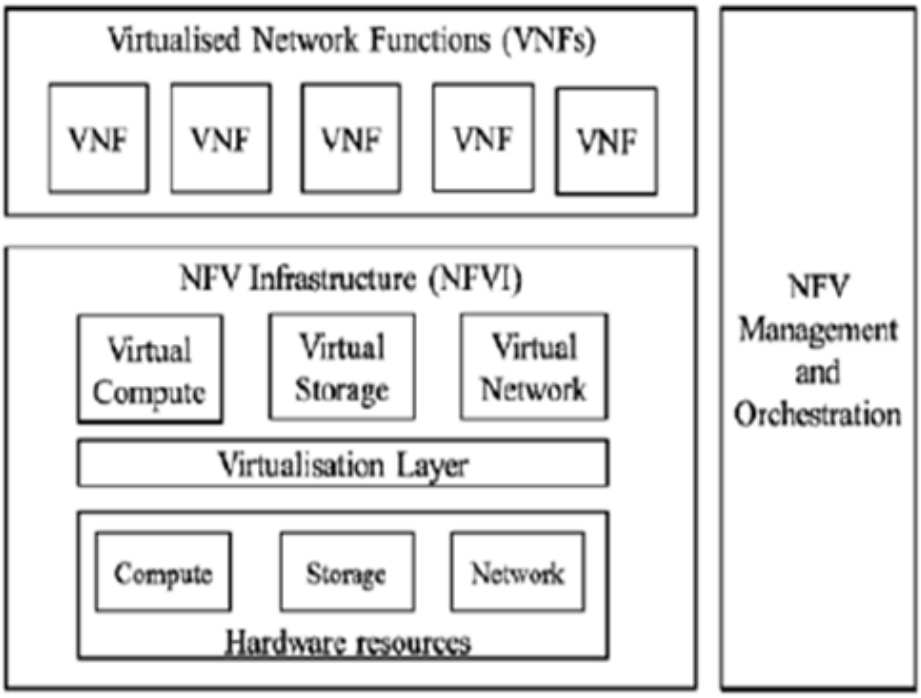

Figure 2-3: High-level NFV Domain [19]

2.2.2 Vendor-Specific NFV Implementations

A variety of NFV implementations have already been deployed by multiple vendors such as CISCO, HUAWEI, HP, ClearWater, Alcatel Lucent, etc. mainly in the telecommunications field. Table 2-1 [20] below describes the developed functionality, infrastructure platform and driving standards used:

	Functionality	Platform	Driving Standards
HP OpenNFV	Open standards-based NFV reference architecture, labs as a sandbox in which carriers and equipment vendors can test vEPC.	OpenStack	ETSI
NFV Open Lab	Supports the development of NFV infrastructure, platforms and services.	OpenStack, OpenDaylight	ETSI
Intel ONP	Provides developers with a validated template for quickly developing and showcasing next-generation, cloud-aware network solutions.	OpenStack, OpenDaylight	3GPP or TMF
CloudNFV	Provides a platform for virtual network service creation, deployment, and management.	OpenStack	TMF and ETSI
Alcatel CloudBand	Can be used for standard IT needs as well as for CSPs who are moving mobile networks into the cloud.	Red Hat Linux OpenStack Platform	ETSI
BroadBand NFV	Migrate virtual functions between platforms based on various vendor solutions.		ETSI
Cisco ONS	Automated service delivery, improved network and data center use, fast deployment of personalized offerings.	OpenStack, OpenDaylight	ETSI
F5 SDAS	Extensible, context-aware, multi-tenant system for service provisioning	OpenStack, BIG-IP, BIG-IQ	IETF, 3GPP, GSMA, ETSI, ONF
ClearWater	SIP-based call control for voice and video communications and for SIP-based messaging applications.	Apache Cassandra, Memcached	3GPP IMS, ETSI TS
Overture vSE	Host multiple VNFs in one box, Accelerate service creation, activation and assurance, Decrease inventory and management costs, Optimize service flexibility, Eliminate trucks rolls	Linux Overture Ensemble OSA , OpenStack	

Table 2-1: Industry NFV Projects [20]

2.2.3 NFV Use cases

Number of Use cases have been defined by ETSI [21], and as NFV scenarios have started being commercially deployed, there might be more cases evolving. Few of NFV use cases related to CDN are as follows.

Virtualization of CDNs (vCDN) [21] motivates the need for deploying cache nodes in Internet Service Provider (ISP) network by CDN provider or content provider rather than using CDN nodes integrated into the operator network for Video Traffic Delivery. This will be highly effective and

cost-efficient way for CDN providers and they can serve content from the nodes nearer to end user with high reliability.

Virtual CDN testing use case [32] aims at testing surrogate server VNFs before deployment. Load tester was set up to generate multiple on-demand client requests for video content, served directly from the cache itself and returned to those clients. This analyzes the effect of various factors such as the use of analytics and the level of granularity of the performance.

Use case on [33] focuses on algorithm aspect of solving cost and resource optimization of surrogate server placement problems. To reduce the network traffic, several virtual CDN (vCDN) caches can be deployed within the network in combination with one or multiple vCDN Request Routers to coordinate the requests. The advantage of additional vCDN Caches not only reduces bandwidth usage but also reduces latency.

Use case proposed on [34] is a value-added service which CDN provider may offer to its customers. Video streaming being the service, a hearing-impaired user can make use of sign language which requires a mixer, transcoder, and compressor VNFs to offer service. Insertion of commercials can be another potential value-added service.

2.2.4 Advantages

There are several advantages in using NFV. Some of them are listed below:

- **Openness of platforms:** By virtualizing the network functions, service providers could avoid the vendor lock-in in their network equipment. Vendor Locked-in has overwhelmed their budgets and degraded their network performance. Adopting openness concept assures

that service providers benefit from multi-vendor network functions that serve their needs [22]. The openness of the platforms would facilitate new revenue trends and opportunities for better contribution and innovation.

- **Scalability and Flexibility:** Virtualized network functions (VNFs) are dynamically scaled to fulfill the traffic demands. VNFs resources could be scaled up in specific locations where demands are high. NFV also allows resources sharing between lightly utilized VNF and higher demanded VNFs. Service providers will benefit from downscaling their VNF resources during off-peak hours. This downscaling allows service providers to benefit from assigning these resources to serve other tasks or can be easily switched off [23].
- **Operation Performance Improvement:** VNFs are installed and initiated automatically by the orchestrator, which senses the network traffic on a real-time basis. Intelligently automated resource allocations or instantiation of new VNFs can improve network service by limiting service interruption by adopting mechanisms such as automatic network failure and fault recovery.
- **Reduced CAPEX and OPEX:** Any upgrade of service requires a long time to develop the software and porting them it to specific hardware, besides the testing and quality assurance. NFV waives all of these concerns and provides the possibility of having the production, testing and development environment running on the same infrastructure. This will lower CAPEX and OPEX investments.

2.3 Domain Name System

In this section, we introduce Domain Name System (DNS), explain its elements such as Domain namespace, resource records, name servers and resolvers and introduce a popular implementation of DNS named BIND server.

2.3.1 A Brief Introduction to DNS

Domain Name System (DNS) is to translate between hostnames and IP address [24]. The DNS is both a distributed database, which is implemented in a hierarchy of name servers and an application-layer protocol which describes how name servers and hosts should communicate. It also provides other details about the domain or host, including reverse mapping from IP addresses to host names and mail-routing information. Since achieving good performance is an important goal of DNS, it makes extensive use of caching. Caching helps in addressing the problems associated with server load and client latency [30].

2.3.2 Main elements in DNS:

The main elements in the DNS topology are:

- Domain name space and resource records – these include the rules for tree structured namespace and the data associated with names respectively [25].
- Name servers – server programs which contain data about the domain tree structure and set information [26].
- Resolvers – these obtain information from name servers in order to answer clients' requests [26].

2.3.2.1 Domain Name Space Structure

The naming convention DNS uses is based on a hierarchical tree structure, called the Domain Namespace. The top node on of the tree is called the root node. Root node includes the top domains, e.g. .com, .org, .net and .se. Organizational domains, which can contain either hosts or additional subdomains, are part of the top domains. Figure 2-4 represents an example of Domain Namespace structure.

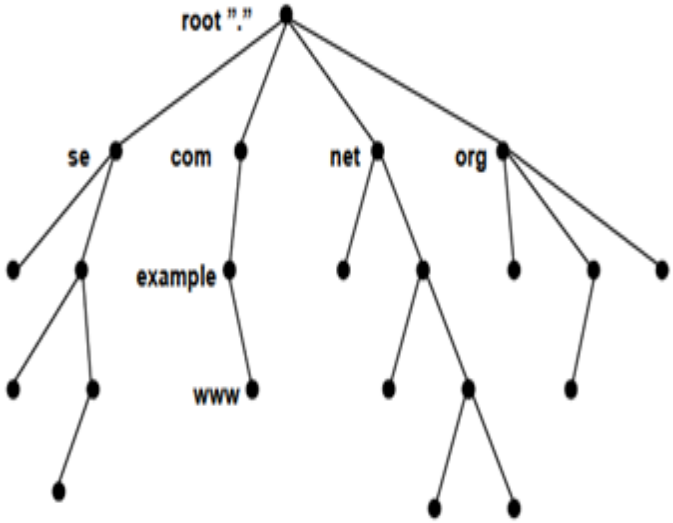

Figure 2-4: The Domain Namespace structure [20]

2.3.2.2 Resource Records

DNS names, IP addresses, and other information are maintained on the DNS servers in the Resource records [27].

- **SOA** (Start of Authority) – This is a required record and there can be only one SOA record in each zone file. The SOA record has information about the primary name server for the zone, the e-mail address to a person responsible for the zone.
- **NS** (Name Server) – Indicates a name server for the zone.
- **A (**Address) – Maps a hostname to an IP-address.
- **CNAME** (Canonical Name) – Maps an alias to its canonical name. If the name server looks up a name and finds a CNAME record, a new lookup is performed using the canonical name.
- **MX** (Mail eXchanger) – Specifies mail exchangers for the zone. Each MX record also has a preference value used to prioritize between different MX records.
- **PTR** (Pointer) – Maps an IP-address to a name.
- **TXT** (Text) – Maps arbitrary and unformatted text with a host.

2.3.2.3 Name servers

The software that handles information about a specific part of the domain namespace is called a name server. The portion of the namespace that the name server has complete information about is called a zone, for which the name server is said to be authoritative.

The single name server cannot handle all the load. So there are three different types of name servers [28] namely local name servers, root name servers and authoritative name servers.

- **Local name servers** – When host issues a DNS query message, the message will initially be sent from the issuing host to the local name server. If the message requests a translation for a host on the same local area network, the name server can send a reply with the requested information instantaneously. If the requested host is located elsewhere and no cached

information about the host is available, the local name server will have to forward the request to other name servers with additional information about the requested hostname.

- **Root name servers** – There are about a dozen of root name servers, which are queried if a local name server does not have a mapping. The root name servers know the addresses of authoritative name servers in the different top-level domains and thus forwards the request to them if not found.
- **Authoritative name servers** – A DNS server is by definition authoritative for a host if it always has the information required to translate hostname into IP-address. Every single host has an authoritative name server, and when a translation request is received for which the name server is authoritative, the name server responds with the requested information.

2.3.2.4 Resolvers

Resolvers [26] should have a direct connection to a minimum of one name server and use its information to answer queries directly or forward the query to other name servers. The resolver is usually a system service which is directly connected to the user programs.

2.3.3 BIND

BIND (Berkeley Internet Name Domain) is a popular implementation of the DNS protocols maintained by the Internet Systems Consortium [29]. BIND is compatible with most UNIX versions and can be ported to Windows environment. BIND is generally used as a testbed for testing new DNS features. It was also the first name server ever implemented. The variety of supported

functions and features and the design is highly modular and cleaner makes the BIND server very popular.

2.4 Summary

In this chapter, we discussed the background concepts which are related to this thesis. First, we introduced the concept of Content Delivery Network (CDN), its architectural components, functionalities types and advantages of CDN. It was followed by a discussion of Network Function Virtualization (NFV), its different domains, various vendor-specific implementations, use cases, and benefits. Finally, we discussed Domain Name System (DNS), its elements and existing implementation of DNS BIND server.

Chapter 3

3. Scenarios, Requirements and State of the Art Evaluation

This chapter consists of four sections. In the first section, we present a motivating scenario. In the second section, we derive the requirements. There are two evaluation sections in our thesis. In the third section, we review and evaluate works on various description models. And in the fourth section, we review various publication and discovery frameworks and evaluate them against our requirements. And in the last section, we summarize the chapter.

3.1 Motivating Scenario

This section presents a motivating scenario in the CDN domain and presents how VNFs can be published and discovered. We start this section by mentioning the early assumptions we made. In the second section, we introduce actors and their interactions related to the scenario. In the third section, we present a scenario in the CDN domain.

3.1.1 Assumptions

Let us consider the case of CDN environment, where there is a CDN provider who is responsible for serving content to end users. Our motivating scenario is based on the following assumptions.

- First assumption is that Content owners pay CDN providers to deliver their content to end-users. So, our CDN providers can distribute the content. CDN providers are also capable of deploying VNF in their infrastructure.
- Second assumption is that VNF providers have already developed VNF.

3.1.2 Actors and their interactions

In our motivating scenario, we have identified five types of actors: End-user, CDN provider, VNF consumer, Repository managers and VNF providers. Each actor plays different roles and we describe them in the following section.

- **End-User***:* The end-user plays the role of consuming content offered by CDN provider.
- **CDN Provider***:* CDN provider serves end-end services to end-user. It has the capability to deploy, instantiate and run VNF.
- **VNF Consumer***:* VNF consumer is responsible for discovering VNFs from repositories depending on metadata published by VNF provider.
- **Repository managers** *:* Repository managers hold VNF descriptors published by various VNF providers. There are several repository managers and each is specific to a VNF provider.
- **VNF Provider** *:* VNF providers are responsible for developing VNF, creating metadata for VNFs and publishing them in their proprietary repository manager. It is to be noted that there are several VNF providers.

Figure 3-1 represents the interactions among different actors in our motivating scenario.

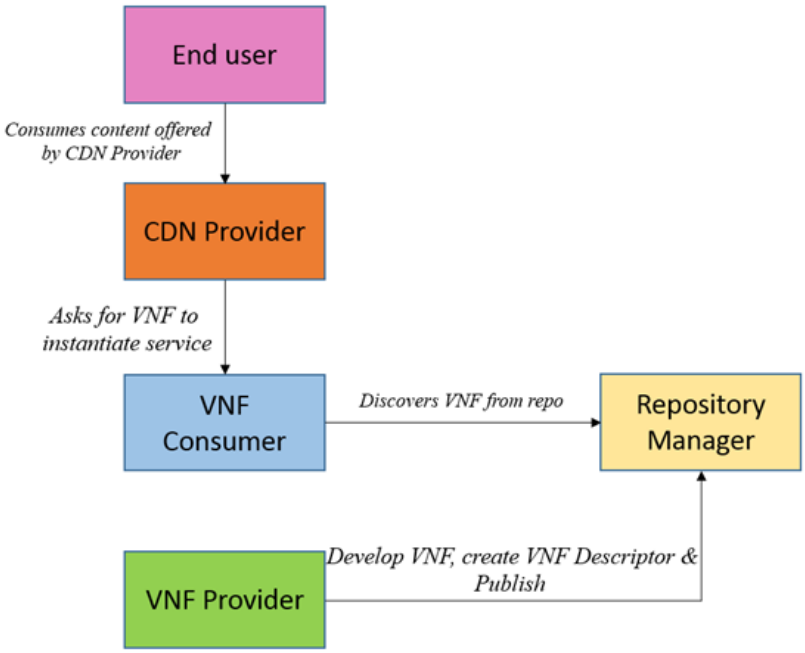

Figure 3-1: Business actors, Roles and Interactions

3.1.3 Value added service provisioning scenario in CDN domain

Virtual Network Functions are gaining popularity in the telecommunications industry and are finding its importance in several application domains. VNF plays an important role in Content Delivery Network applications since many value-added services can be offered to end-users on demand. Publication and discovery of VNFs are crucial for provisioning any value-added services. We now present a motivating scenario in CDN domain.

Let us consider an end user who is interested in watching a video. End-user requests video from CDN provider. Video content is available in mp4 format with CDN provider and it sends it to end user. Say, the end-user mobile device does not support mp4 format and supports only AVI format.

CDN provider is responsible for serving the content to end user in AVI format to avoid any intervention. Transcoder VNF has been already developed and published by VNF provider. Now, CDN Provider has the responsibility to discover the location and other characteristics of transcoder VNF, so that he can download and execute them to offer service. So it requests the VNF consumer, to search for the most relevant VNF.

Let us consider there are multiple VNF providers. They develop VNFs such as Firewall, Transcoders, Network Address Translation devices, etc. and publish the descriptors to their corresponding repository managers. Consider, VNF provider 1 developed firewall and mixer VNF and published its descriptors along with its location details in Repository manager 1. Similarly, VNF provider 2 developed NAT and transcoder VNF and published their details. Now, repository manager 2 has NAT and Transcoder descriptor.

VNF consumer, on behalf of CDN provider request tries to contact repository managers and reads the descriptors from them one by one. If a transcoding service descriptor which matches the required characteristics are found, then VNF consumer would inform the CDN provider. VNF consumer sends the URL, where the description and transcoder VNF is available. Then CDN provider using the provided URL, connect, fetch, download and instantiate them in its premises. It then makes use of the service and transcodes video to AVI format. Once successfully transcoded, CDN provider serves the end user requested video in AVI format. Figure 3-2 provides an illustrative diagram for interaction among the actors in value-added service provisioning scenario.

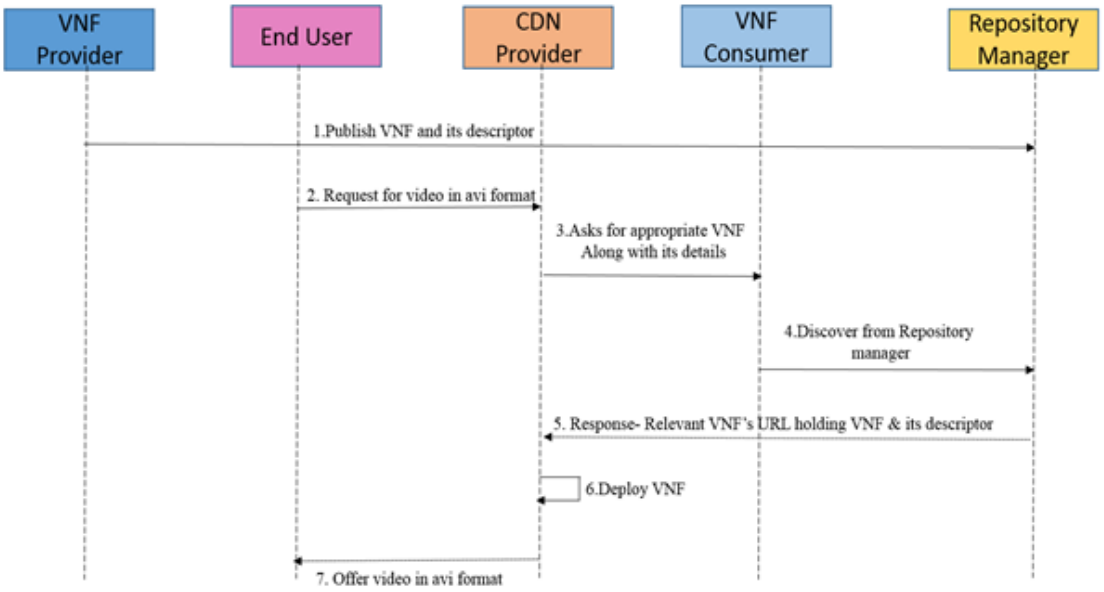

Figure 3-2: Interaction among the actors in Value-added Service Provisioning Scenario

3.2 Requirements

We have derived requirements based on our motivating scenario which is described in the previous section. We have classified them into two sections. The first section presents general requirements derived from our motivating use case. The second section represents requirements more specific to publication and discovery mechanism.

3.2.1 General Requirements

The following three general requirements are derived.

1) Caching: The first need is to enable caching. VNF consumer receives a request from service provider every time when a new service has to be instantiated. There is a necessity to refer repository manager whenever a request has been received. Caching the VNF details by VNF consumer will significantly reduce the roundtrip delay.

2) Geographically distributed: Requests can be invoked by an end-user from any part of the world. When VNFs and its descriptors are geographically distributed, service providers can instantiate VNFs close to the end-user location. This will eventually reduce latency in serving value-added services.

3) Based on industry standards: The proposed solution for publication and discovery mechanism has to be leveraged on industry standards.

3.2.2 Publication and discovery requirements

We derive four requirements here. The first requirement is common for VNF providers and VNF consumers, second and third requirements corresponds to VNF providers and VNF consumers respectively.

4) Universal repository for VNF publication and discovery: VNF providers and VNF consumers need a common repository to publish the descriptors of developed VNFs and to discover them respectively. Federated registry to publish VNF descriptors offered by multiple VNF Providers will have a massive impact in discovery. Also, this can help VNF providers to advertise all the offerings available. VNF consumers upon request by a service provider can search for services by going through the published descriptors available in a unified repository, rather than approaching proprietary repository managers published by VNF Providers.

5) *Unified and common model of VNF descriptors*: Our fifth requirement is the necessity to describe VNFs offered by various VNF providers using a common structure. This common structure has to be exhaustive so that it is inclusive of all necessary details and flexible enough to add any further details. Also, VNF descriptor should be in human readable and understandable format, so that VNF consumers will spend less time in decrypting the necessary details of relevant VNFs upon the use of a unified structure. Also, there is a need for VNF providers to describe VNFs in a standard and pervasive format.

6) *Automatic and reliable discovery procedures:* Our sixth requirement is the introduction of discovery procedures for matchmaking the VNF request from a service provider and published VNF metadata offered by VNF provider. These procedures should be functioning automatically, i.e., whenever a request is placed, VNF consumer should start the discovery process. VNF thus discovered should also be relevant and reliable, i.e., VNF consumers should figure out the most appropriate VNF. When there are two transcoders available in the repository, this discovery procedure should be able to select most appropriate VNF based on the requirements.

3.3 Evaluation of Description models

In this sub section, we are going to evaluate three well-known description models such as Web Service Description Language - WSDL, Web Application Description Language – WADL, and Yet Another Markup Language – YAML.

Authors in [51] describes WSDL, as a collection of communication end points that can exchange certain messages to describe Web services. WSDL is developed by IBM and Microsoft and is World Wide Web Consortium (W3C) recommended. WSDL document exposes operations, which

parameters to pass to a specific operation, protocol via operation can be accessed and location where wed service resides. Broadly classified, WSDL service description holds two main pieces of information: (i) an application level service description, or abstract interface and (ii) protocol-dependent details users follow to access endpoints.

Abstract description has three main components: vocabulary, message and interaction. WSDL uses external type systems, mostly XML Schema definition - XSD for information exchange. Messages provide an abstract, typed data definition sent to and from the services. *message* element includes *parts*, which describes *XSD* types or elements from previously used vocabulary. Interaction consists of *operation* and *portType*. *operation* indicates what part a particular message plays in interaction. It can be *input*, *output* or *fault*. *portType* can be defines as a collection of operations that are supported by an endpoint. All these elements are used to describe a service's application-level functionality. By using *documentation* element inside any other WSDL elements, we can make the WSDL document human-readable.

Concrete Binding Information addresses details on what communication protocol to use, how to achieve service interaction over protocol, and where to terminate communication. *binding* element mentions SOAP when used, and be specific on interaction type such as Remote Procedure Calls – RPC, or encoding which uses XSD to describe the transmitted XML. *port* element describes a single end point that processes the request. Figure 3-1 gives a complete depiction on WSDL's abstract description and concrete binding information.

Web services are by design highly extensible [55] and therefore their descriptions too. WSDL documents can either be kept either very simple, using the basic elements and data-types from XML schema, or in a modular manner, distributed to any extend and using self-defined data types. Thus

WSDL documents can be very well used for describing VNFs in an exhaustive manner. Figure 3-3 provides an example of WSDL document.

```
<definitions name="HelloService"
   targetNamespace="http://www.examples.com/wsdl/HelloService.wsdl"
   xmlns="http://schemas.xmlsoap.org/wsdl/"
   xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
   xmlns:tns="http://www.examples.com/wsdl/HelloService.wsdl"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema">

   <message name="SayHelloRequest">
      <part name="firstName" type="xsd:string"/>
   </message>

   <message name="SayHelloResponse">
      <part name="greeting" type="xsd:string"/>
   </message>

   <portType name="Hello_PortType">
      <operation name="sayHello">
         <input message="tns:SayHelloRequest"/>
         <output message="tns:SayHelloResponse"/>
      </operation>
   </portType>

   <binding name="Hello_Binding" type="tns:Hello_PortType">
      <soap:binding style="rpc"
         transport="http://schemas.xmlsoap.org/soap/http"/>
      <operation name="sayHello">
         <soap:operation soapAction="sayHello"/>
         <input>
            <soap:body
               encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
               namespace="urn:examples:helloservice"
               use="encoded"/>
         </input>

         <output>
            <soap:body
               encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
               namespace="urn:examples:helloservice"
               use="encoded"/>
         </output>
      </operation>
   </binding>

   <service name="Hello_Service">
      <documentation>WSDL File for HelloService</documentation>
      <port binding="tns:Hello_Binding" name="Hello_Port">
         <soap:address
            location="http://www.examples.com/SayHello/" />
      </port>
   </service>
</definitions>
```

Figure:3-3: Example of WSDL document [51]

Authors in [52] describe Web Application Description Language (WADL) as machine process-able protocol description format to use with HTTP based Web Applications, mostly with XML. WADL was submitted to W3C by Sun Microsystems but has not been standardized [54].

All WADL elements has an XML namespace and most of the elements are extensible using additional attributes or child elements from another name space. WADL document has following components.

Application element is the root element, which includes *grammars* and *resources* element which are optional, and zero or many of elements such as *method, representation and fault*.

grammars acts as a container for definitions of XML structures exchanged during protocol execution. It has one child element called *include*, which allows definition of one or more XML structures to include by reference attribute. Include element has *href* attribute of type *xsd:anyURI* where *href* specifies an URI

resources elements consists of all resources provided by the application. It has a *base* attribute of type *xsd:anyURI* which specifies base URI of child resource identifier. It has a child element *resource,* which describes a single resource and has an *uri* attribute. It consists of following child elements: an optional *path_variable*, zero or more *method* and *resource*. *path_variable* is used to parameterize the identifier of parent *resource* and has *name* and *type* attribute of type *xsd:NMTOKEN* and *xsd:string* by default respectively.

method element describes input and output to HTTP protocol to access a *resource*. It has one of either attributes. A) *name* and *id* which specifies http method and identifier or B) *href* which is used to reference a method somewhere using *id* attribute. *method* has two child elements, *request* and *response*. *request* specifies input to the method and *response* specifies output of the method. *request* has no attributes but may contain following child elements such as zero or many *representation* elements

and *query_variable* elements. *query_variable* represents a URI query parameter and has no child elements but with attributes *name, type, required, repeating and fixed. response* element has two child elements namely *representation* and *fault.*

representation element describes state of a resource and can be declared globally as a child of *application* element, locally as a child of *request* and *response* or referenced externally. It has one of the following combination of attributes. A*) id, mediaType and element* or b) *href.* It can have zero or more child *representation_variable. representation_variable* is used for parameterization and has no child elements but with attributes *namely name, type, path, required, repeating* and *fixed*.

fault element denotes an error condition and is not mandatory. It has one of the following combination of attributes. A*) id, mediaType and element* , b) *href* or c)*status* . *status* attribute gives information on HTTP status code when an error happens. Fault variables are a direct child element of *fault* variables. Figure 3-4 shows an example of simple WADL document structure.

```
<application xmlns="http://research.sun.com/wadl"
  xmlns:aws="http://webservices.amazon.com/AWSECommerceService/2005-07-26"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <grammars>
    <include href="AWSECommerceService.xsd"/>
  </grammars>

  <resources base="http://webservices.amazon.com/onca/">
    <resource uri="xml">
      <method href="#ItemSearch"/>
    </resource>
  </resources>

  <method name="GET" id="ItemSearch">
    <request>
      <query_variable name="Service" fixed="AWSECommerceService"/>
      <query_variable name="Version" fixed="2005-07-26"/>
      <query_variable name="Operation" fixed="ItemSearch"/>
      <query_variable name="SubscriptionId" type="xsd:string"
        required="true"/>
      <query_variable name="SearchIndex" type="aws:SearchIndexType"
        required="true"/>
      <query_variable name="Keywords" type="aws:KeywordList"
        required="true"/>
      <query_variable name="ResponseGroup" type="aws:ResponseGroupType"/>
    </request>
    <response>
      <representation mediaType="text/xml"
        element="aws:ItemSearchResponse"/>
    </response>
  </method>
</application>
```

Figure 3-4: Example of WADL document [52]

Authors in [53] described YAML Ain't Markup Language as a human-friendly data serialization language. It is not a registered Internet media type. Data is presented in a meaningful way and represents cleanness by minimizing the number of structural characters. It is expressive and extensible depending on needs, which gives the possibility for describing VNF exhaustively. Authors in [19] mentioned although it is human readable, it is not human understandable. Another disadvantage is that it is very difficult to parse. YAML can be considered as a superset of JSON. JSON is easy to generate and parse but is not as human-readable as YAML. XML is primarily

designed to support structured documentation and has more design constraints when compared to YAML.

YAML uses indentations for scope, block sequences indicates entry with dash and space (“- “), mappings use colon and space (“: “) for a key-value pair, and comments use hash (“#”). Flow

```
tosca_definitions_version: tosca_simple_profile_for_nfv_1_0_0
metadata:
    template_name: sample-tosca-vnfd-for-vnfc

topology_template:
  node_templates:
    firewall_vnfc:
      type: tosca.nodes.nfv.VNFC.Tacker
      requirements:
        - host: VDU1
      interfaces:
        Standard:
            create: install_vnfc.sh

    VDU1:
      type: tosca.nodes.nfv.VDU.Tacker
      properties:
        image: fedora-software-config
        flavor: m1.small
        mgmt_driver: noop
        key_name: stack_key
        config: |
          param0: key1
          param1: key2

    CP1:
      type: tosca.nodes.nfv.CP.Tacker
      properties:
        management: true
        anti_spoofing_protection: false
      requirements:
        - virtualLink:
            node: VL1
        - virtualBinding:
            node: VDU1
    VL1:
      type: tosca.nodes.nfv.VL
      properties:
        network_name: net1
        vendor: Tacker
```

Figure 3-5: Example of YAML template [53]

sequence is denoted by separating list by commas within square braces (“,”) and flow mapping uses curly braces. It uses three dashes (“---“) to separate directives from document content. Three dots “…” represents the end of the document. Repeated objects are first identified by ampersand “&” and are further referenced by asterisk “*”. Question mark and space “? “ indicate complex mapping key and key value pairs can be represented soon after dash, colon, or question mark. Scalar content can be written in block notation using literal style “ | ” or folded style using “>”. Tags can be of many types such as int, float, null, binary, omap, set, seq, mao, str, etc. Explicit typing is also possible with a tag using the exclamation point (“!”). Figure 3-5 shows an example of sample YAML VNF template.

Table 3-1 represents a table on evaluating three existing description models based on our requirements specific to VNF descriptors.

Related Work Requirements	WSDL	WADL	YAML
Based on industry standards	Yes	No	No
Exhaustive way of describing VNFs	Yes	Yes	Yes
Human readable and human understandable	Partially	No	Partially

Table 3-1: Evaluation of Description Models

3.4 Evaluation of Publication & Discovery Frameworks

Recent researches on NFV mainly focuses on service chaining, management, and orchestration operations. Although few work is done on publication and discovery aspect of VNFs, there are more aligned towards deployment perspective. VNFs gaining its importance and affecting CAPEX and OPEX costs majorly, provisioning a service is crucial. For provisioning services on demand, publication and discovery of VNF are of utmost importance.

Currently, there are several ways to describe VNFs in the market. But, most of them are described considering only their deployment characteristics. For instance, OpenMANO [46] project has been contributed to the open source community project Open Source MANO (OSM),

hosted by ETSI. Openmano descriptors used to define VNFs describe them in such a way to deploy it appropriately.

Open Baton [47] is another open source project providing an implementation of the ETSI Management and Orchestration (MANO) specification. This uses the descriptor proposed by ETSI MANO specification. ETSI [48] mentions VNFD proposed by them as a deployment template, which describes VNF in terms of deployment and operational behavior. OpenBaton project is capable of handling VNF descriptors in TOSCA [49] template. This describes VNF using a service template and node templates. These include details more concerned to its virtual links and capabilities, considering their deployment behavior.

NFV proposed YAML descriptors are human-readable but are not human-understandable [50]. Network Modelling (NEMO) [50] language aims at proposing VNF, which are humanly understandable. These again define NodeModels which is used to instantiate in the infrastructure, which falls into the previous group of deployment-specific description.

In this section, we present current state-of-the-art solutions similar to our research area and evaluate them critically. We classify them into two groups. The first group involves the research work on VNF publication and discovery. In the second group, we consider state of the art solutions on general frameworks available for publication and discovery. After reviewing the state of the art solutions, we evaluate them based on our requirements.

3.4.1 VNF Publication and Discovery Frameworks:

Authors in [35] propose a tool named VNF Onboarding Automation Tool (VOAT) to onboard new services. VOAT does not include any caching mechanism for future discovery, which does not meet

our first requirement. This tool does not talk about the geographical distribution of VNF descriptors and efficient discovery processes close to end-user locations but can be extended for the same. So, our second requirement is partially discussed. This uses ETSI VNFD, Parser from Openstack Heat Translator Project, Service catalogs such as Comptel catalog and IBM UCD Blueprint. REST and CLI are used for interfacing between VOAT and external software which satisfies our third requirement. This tool accepts metadata and artifacts provided by vendor input table and convert them into HOT template for future instantiation. They store these templates in persistent asset manager repository to create VNF. Once created, they publish VNF in the service catalog, ready to onboard and deployment. Although Asset manager holds artifacts and metadata, the same repository is not used for any discovery purposes, so our fourth requirement is partially satisfied. Descriptors are created based on user input and it does not provide details about the structure or format. So, our fifth requirement is not met. There are no discovery mechanisms associated with is work which leaves our sixth requirement not met.

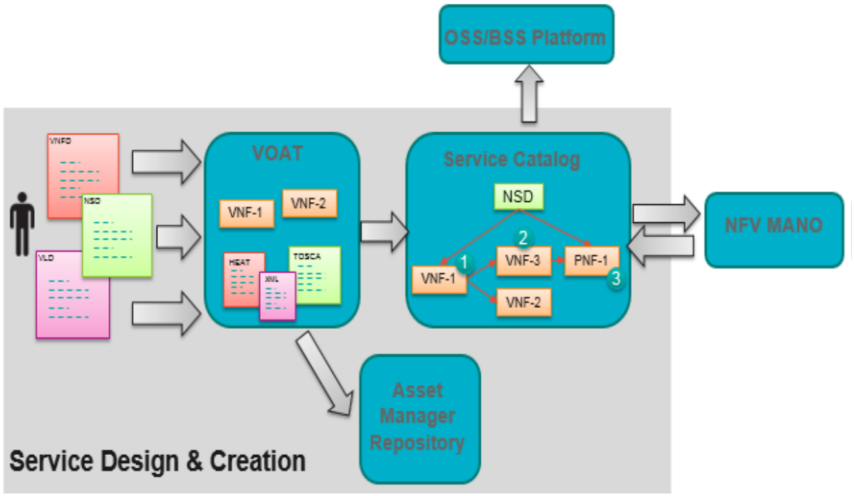

Figure 3-6: VOAT Architecture [35]

[36] propose a marketplace for the users to trade, auction, select and acquire NFV services. Brokerage module does not include any caching mechanisms which leave our first requirement not met. This marketplace can be used worldwide but there are no mechanisms associated with providing results with minimal latency. So, our second requirement is partially satisfied. There is no discussion on our third requirement which is information on the standards it followed. NF store is considered as a hub where VNFs are available, but there is no information on how they are published and whether they are used for discovery. So our fourth requirement is partially satisfied. It does not provide any information of VNF descriptors and interfaces among different module, and hence our fifth requirement is not addressed. Customers select the offerings but it does not include any discovery procedures, so our sixth requirement is not met.

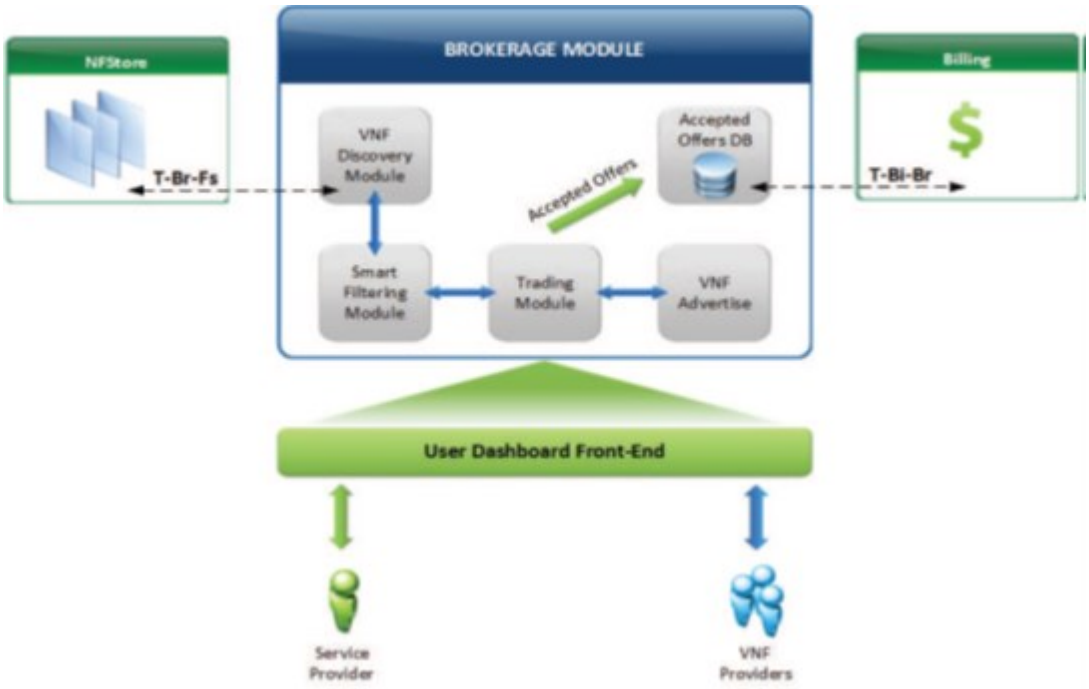

Figure 3-7 : Brokerage module internal architecture [36]

Authors in [37] introduce a concept which allows operators not only to deploy VNFs but also to offer them to customers as value added services. T-NOVA does not include any caching feature, so our first requirement is not met. This can be used universally for discovery purposes but there is no mechanism involved for minimal latency, thus our second requirement is partially satisfied. T-NOVA follows ETSI standards, uses open API such as OpenStack's RESTful API and Open Cloud Computing Interface (OCCI) which meets our third need. This introduces NFV marketplace where VNFs can be published, acquired and instantiated. Network Function Store is introduced which contains VNF and its metadata, which can be used for discovery purposes also. This satisfies our fourth requirement. It fails to discuss the descriptor structure/format, and hence our fifth requirement is not discussed. Discovery of VNFs is again based on selecting procedure, so our sixth requirement is not met.

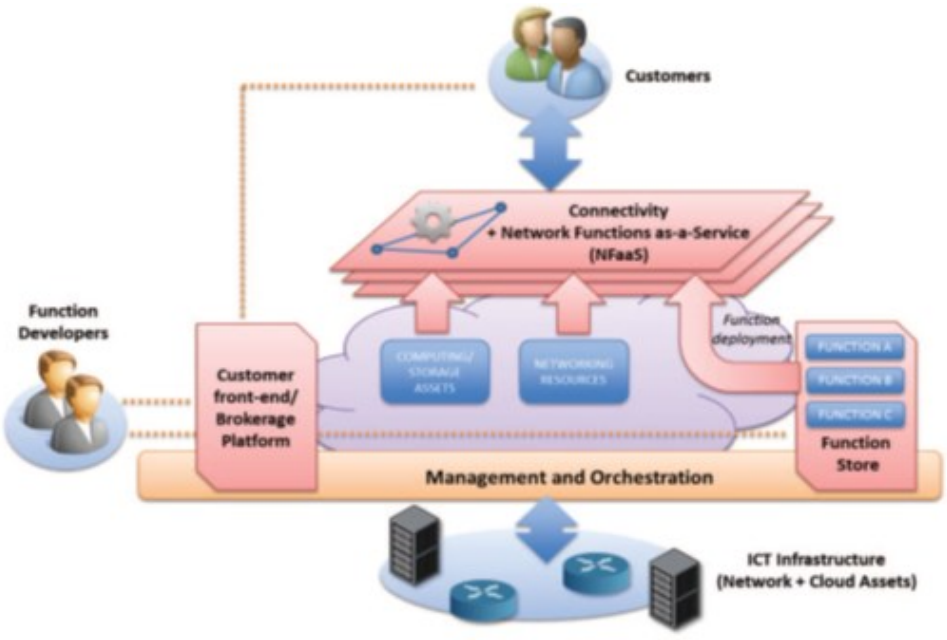

Figure 3-8: High level visualization of T-NOVA architecture [37]

OpenBaton [38] is another implementation of ETSI MANO specification. OpenBaton does not include any component for enabling caching. So our first requirement is not met. OpenBaton can be used universally but there is no mechanism included for delivering the VNF at minimal latency rate. This leads to the partial satisfaction of our second requirement. OpenBaton follows ETSI Management and Orchestration (MANO) specification, REST and CLI interfaces are used for communication among various components which meets our third requirement. This provides Network Function Virtualization Orchestrator NFVO for managing virtual network functions. VNF Descriptors are not stored in any repository and there is no federated repository used during the discovery process. Hence, our fourth requirement is not met. VNF descriptors are defined in YAML format considering essential properties of VNF. This meets our fifth requirement. The customers are allowed only to select from the available VNFs and it does not include any discovery procedures against its requirement, so our sixth requirement is not met.

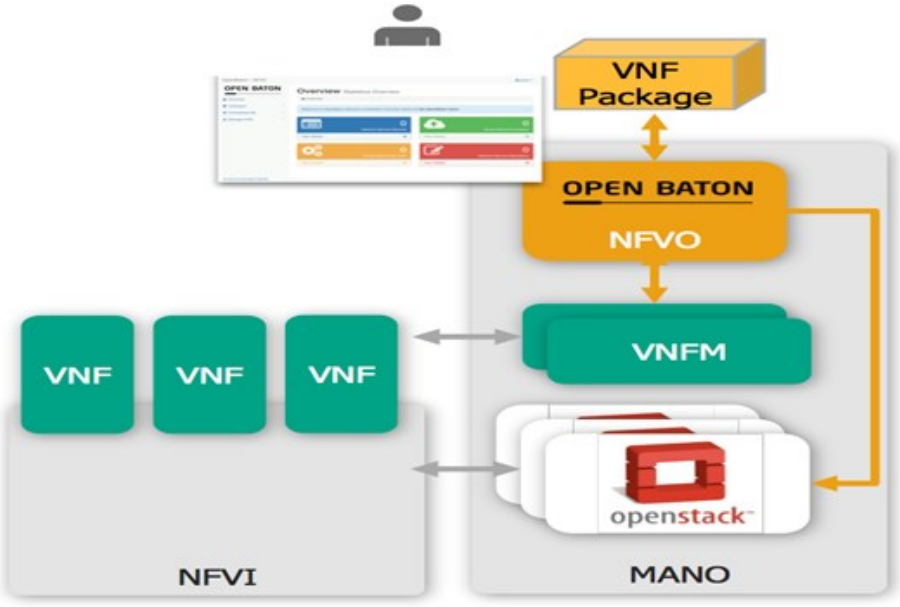

Figure 3-9: OpenBaton Architecture [38]

3.4.2 Publication and Discovery Frameworks:

In this section, we evaluate existing works on publication and discovery framework.

Authors in [39] propose a new Operational model with an entity named Service Invocation Proxy (SIP), to enable service brokers in Service Oriented Architecture (SOA) to play an important role. Introduction of this can overcome the bottlenecks associated with Universal Description, Discovery, and Integration (UDDI) registry's single point of failure. SIP has the role of mediating service calls between service producers and service consumers. This component aids in enriching service descriptions, which will eventually enhance service discovery and selection. SIP caches the request pattern and response of service consumers and service provider's thus reducing response time and cost. This satisfies our first requirement. This work did n0t address the global distribution of services, leaving our second requirement unfulfilled. SOAP/HTTP interfaces are used for communication among various components. This work is implemented through Universal Web service (UWS), which is not popular and industry standard, so our third requirement is only partially met. Service Registry is the universal repository for holding all the descriptors published by Service providers which satisfy our fourth requirement. This operation model does not talk about service description model and our fifth requirement is left undiscussed. Also, Service Invocation Proxy acts a middleman for discovering services by Service consumer whenever a request is instantiated. This meets our sixth requirement.

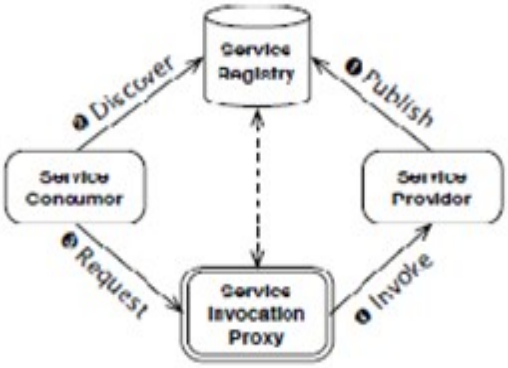

Figure 3-10: Extended Diamond SOA Operational Model [39]

Authors in [40] introduce a discovery mechanism based on UDDI's agent and Directory Facilitator (DF) ontology based on the standard of Foundation for Intelligent Physical Agents (FIPA). This work proposes a use case on how effective the discovery process will be while using the combined UDDI and DF. When service target is unclear, agent queries DF to get appropriate Ontology agent list and refers to corresponding ontology. Then the agent will perform a further search in UDDI. This mechanism will also be useful when the service target is clear but not found. There is no caching mechanism involved in this procedure which leaves our first requirement unfulfilled. Also, this work does not address how the services and descriptions can be distributed. FIPA is an IEEE Computer Society standards which is gaining sharp focus. UDDI registries stopped updating since past few years and hence our third requirement is only partly satisfied. UDDI registries are used for publication of resources. Thus our fourth requirement is met. Agent Communication Language (ACL) is used for DF registration but it does not talk about the service description model, thus our fifth requirement is partially met. Web Service discovery Mechanisms lacks the of web service content and hence is not automated, which fails to meet our sixth requirement.

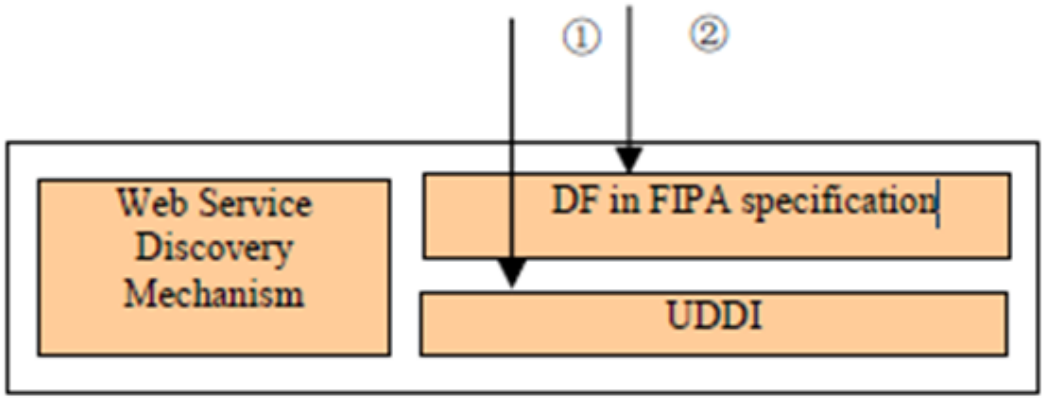

Figure 3-11: Model of Web Service Discovery mechanism [40]

Authors in [41] propose a framework named CatalogNet, which focus on service description model and service discovery mechanism, Services in CatalogNet uses XML based description format and considers static and dynamic contexts. Directory Server (DS) maintains all the distributed systems and client programs connects to these. DSA is the directory server admin which holds the topology and forwards status, which basically cache the high-level details and hence our first requirement is partially satisfied. Backup DSA is also available. Service Provider client and Service request client is provided to service providers and service requesters for managing the registered services and for submitting queries respectively. There is no discussion about the geographical spread of services or descriptions which is our second requirement. There is a discussion on standard interfaces, such as HTTP between DS. This also adopts client/multi-server architecture which is the industry standard and thus it meets our third requirement. DS holds all the service related information, thus our fourth requirement is met. Service Description Model is XML based and has a unified and common model for representing services, thus our fifth requirement is met. Service index algorithm is used to aid in discovery which meets our sixth requirement.

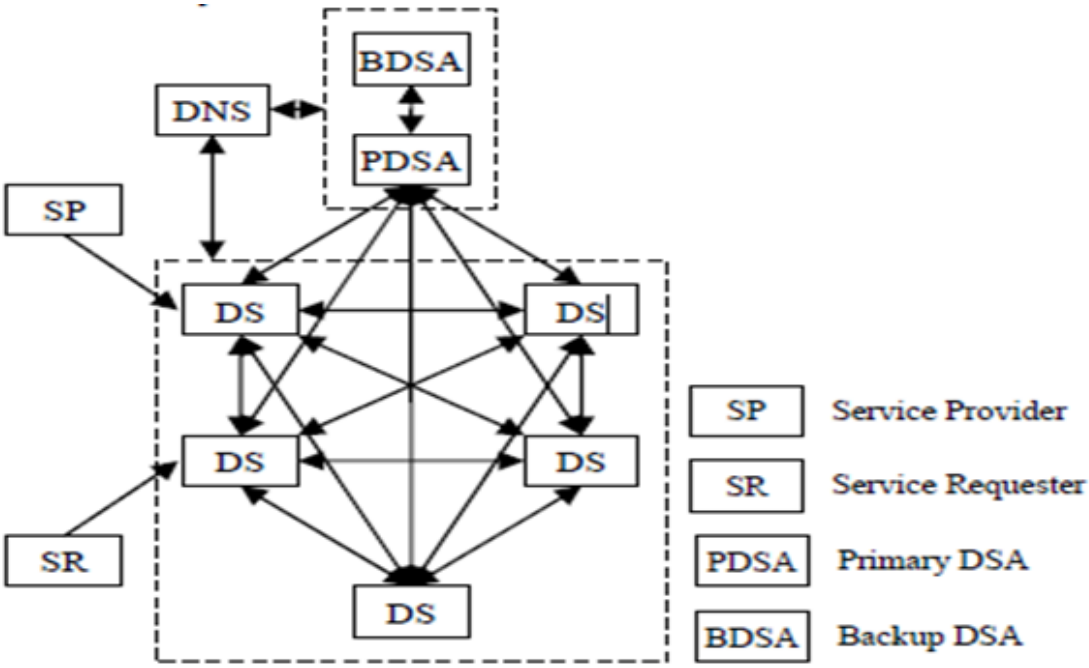

Figure 3-12: System Topology [41]

Authors in [42] propose a three-layer structure of distributed UDDI which overcomes single point of failure and imbalance access associated with centralized UDDI approach. This interoperable model includes UDDI-ROOT, subsidiary UDDI, and registered users. UDDI-ROOT Servers as root UDDI does not provide Web Services with publication and inquiry functions. They mainly record all relevant information of subsidiary UDDI and provides functions of registration and inquiry for the subsidiary UDDI. Since they cache the information of subsidiary UDDI, our first requirement is met. Although subsidiary UDDI composing UDDI-SUBn Server and a UDDI-SUBn clients are mainly responsible for service publication and discovery by end-user, they are not geographically distributed, leaving our second requirement not satisfied. This approach follows UDDI, which is not currently being updated and hence our third requirement is partially met. This does not have a universal repository for publication and

discovery purposes and no description model is proposed, hence our fourth and fifth requirements are not met. This uses find API function of UDDI, meeting our sixth requirement.

Figure 3-13:The design of distributed UDDI system architecture [42]

Authors in [43] summarize Web Application Description Language (WADL) as a machine process-able protocol description format to use with HTTP applications. Fig 3-14 depicts the WADL for Amazon item search service [44] application. Since there is no caching feature introduced, our first requirement is not met. The service is described using a set of resource elements. Each resource contains param elements to describe the inputs and method elements which describe the request and response of a resource. The request element specifies how to represent the input, what

types are required and any specific HTTP headers that are required. The response describes the representation of the service's response, as well as any fault information, to deal with errors. There is no discussion on the geographical distribution of service description, and mechanisms to discover them with minimal latency. WADL has a predefined structure to describe but it is not industry standard, hence our third requirement is not met. There is no common repository for publication and discovery and hence our fourth requirement is not met. WADL has a predefined structure to describe functionalities and hence our fifth requirement is met. No automatic or discovery procedures are used here, thus leaving our sixth requirement not met.

```
<application xmlns="http://research.sun.com/wadl"
  xmlns:aws="http://webservices.amazon.com/AWSECommerceService/2005-07-26"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <grammars>
    <include href="AWSECommerceService.xsd"/>
  </grammars>

  <resources base="http://webservices.amazon.com/onca/">
    <resource uri="xml">
      <method href="#ItemSearch"/>
    </resource>
  </resources>

  <method name="GET" id="ItemSearch">
    <request>
      <query_variable name="Service" fixed="AWSECommerceService"/>
      <query_variable name="Version" fixed="2005-07-26"/>
      <query_variable name="Operation" fixed="ItemSearch"/>
      <query_variable name="SubscriptionId" type="xsd:string"
        required="true"/>
      <query_variable name="SearchIndex" type="aws:SearchIndexType"
        required="true"/>
      <query_variable name="Keywords" type="aws:KeywordList"
        required="true"/>
      <query_variable name="ResponseGroup" type="aws:ResponseGroupType"/>
    </request>
    <response>
      <representation mediaType="text/xml"
        element="aws:ItemSearchResponse"/>
    </response>
  </method>
</application>
```

Figure 3-14 : WADL Description for Amazon item search [44]

Authors in [45] propose a new service discovery model where the quality of service is taken as constraints when searching for Web services. There are four components such as Web service supplier, Web service consumer, Web service QoS certifier, and the new UDDI registry. Web service provider needs to supply information on functional aspects and quality of service. It communicates with its QoS certifier and then registers upon success in UDDI registry. During discovery, Web service consumer takes QoS as a constraint and deliver WSDL files. QoS certifier has details on service provided by Web service supplier and thus our first requirement is met. It does not talk about geographical distribution of UDDI registries and mechanisms for delivering at minimal latency. So our second requirement is not discussed. It uses UDDI registry where 48% links are unusable, which meets our third requirement partially. UDDI registry is a repository for publication of services and discovery and hence it meets our fourth requirement. They propose a model with five data structure types such as businessEntity, businessService, bindingTemplate, publisherAssertion and tModel which meets our fifth requirement. Web Service Consumer once discovers the description has to invoke Web service Supplier on own, which states discovery procedure is not automatic. Hence our sixth requirement is not met.

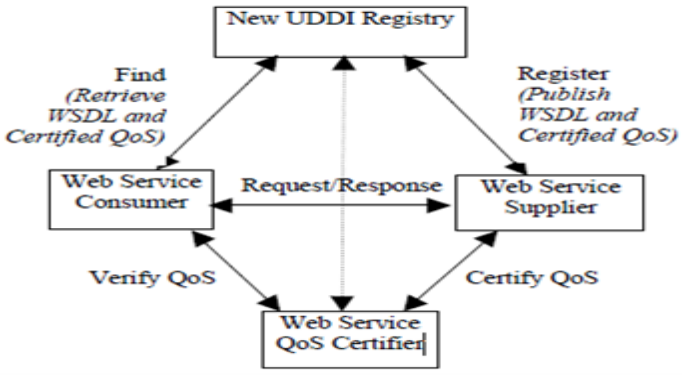

Figure 3-15: Web Service Registration and Discovery Model [45]

Table 1-1 summarizes evaluation of publication and discovery frameworks. To the best of our knowledge, there is no work which fulfils our requirement completely.

Requirements Publication & Discovery Frameworks	**Caching**	**Geographical Distribution**	**Industry standards**	**Universal Repository for Publication & Discovery**	**Unified & Common model**	**Automatic, reliable and precise discovery procedures**
[35]	No	Partially satisfied	Yes	Partially satisfied	Not addressed	No
[36]	No	Partially satisfied	Not addressed	Partially satisfied	Not addressed	No
[37]	No	Partially satisfied	Yes	Yes	Not addressed	No
[38]	No	Partially satisfied	Yes	No	Yes	No
[39]	Yes	Not addressed	Partially satisfied	Yes	Not addressed	Yes
[40]	No	Not addressed	Partially satisfied	Yes	Partially satisfied	Yes
[41]	Partially satisfied	Not addressed	Yes	Yes	Yes	Yes
[42]	Yes	No	Partially satisfied	No	No	Yes
[43]	No	Not addressed	No	No	Yes	No
[45]	Yes	Not addressed	Partially satisfied	Yes	Yes	No

Table 3-2: Summary of Evaluation of Publication & Discovery Frameworks

3.5 Chapter Summary

In this chapter, first, we presented the motivating scenario for VNF publication and discovery. Then we derived the general requirements on scenario and requirements related to publication and discovery in the second section. In the third section, we reviewed various description models. And in the fourth section, we reviewed and evaluated the various architecture related to publication and discovery frameworks. And finally, we found that none of the existing solutions satisfies our requirements completely.

Chapter 4

4. Proposed Architecture

In the third chapter, we derived a set of requirements and evaluated various descriptor models and Publication and discovery framework. In this chapter, we propose description model for VNFs and an architecture for a VNF publication and discovery based on our requirements. We start by introducing VNF descriptors and VNF descriptor model. Then we propose our VNF descriptor using an existing model used in our value-added service provisioning use case. Followed by the descriptor model, we focus on overall architecture for publication and discovery of VNFs. Then, for value added service provisioning, we describe an illustrative scenario for VNF publication and discovery. Next, we discuss how the requirements are met by the architecture. Finally, we summarize this chapter.

4.1 Introduction

In this section, we first focus on introducing the VNF descriptor and its importance on publication and discovery aspects. Then we propose our VNF description model.

4.1.1 VNF Descriptor

VNFs are developed by several developers and provide them with a descriptor. Since VNF Providers are numerous and widespread in various locations, they come in a variety of structure and format. It is essential to have a well-defined descriptor file since they play the key role in publication and discovery aspects. For effective and efficient discovery procedures, and for a federation of VNFs, there is a need for an exhaustive description of VNFs. VNF descriptor should

include details not specific to deployment but also consider its properties and functionalities. Descriptor file provided when includes details about its location will help VNF consumer in downloading and executing them upon discovery.

4.1.2 VNF Descriptor model

There are few key principles for VNF descriptors.

(1) They should be human readable and understandable

(2) Should follow industry standards

(3) Should be exhaustive and extensible

Based on our key principles and already existing descriptor models, we chose WSDL as our description model to describe our VNFs. Our proposed WSDL descriptors are W3C recommended. It is human readable making it easier for any consumer to understand them well. WSDL descriptors are also industry acceptable standard, which allows us to describe VNFs using them. It is possible to describe our VNFs in a comprehensive way using WSDL which also provides flexibility for VNF providers to include or remove details whenever necessary.

Proposed VNF descriptor is mainly based on VNFs properties and operations. Properties can be of two types, functional and non-functional. Similarly, operations are also of two types namely Management operations and Execution operations. Every VNF descriptor has an endpoint which specifies the location where VNF provider has stored them, and are unique. Functional properties specify the details concerning the capabilities and requirements including its type, access specifier, size, vCPU, threshold, etc. Similarly, non-functional properties include details specific to VNF

provider, billing details, size, cost, execution environment details, etc. Execution operation includes inputs and output details concerned to VNF execution. Similarly, management operation specifies the corresponding input to be provided to the input and the expected output after the operation on VNF.

4.1.3 Proposed VNF Descriptor Model

In this section, we propose our VNF descriptor model which we have used in our use case. Transcoder offered by a VNF provider has been described considering its properties and operations. VNF descriptors are exhaustive and describe various details about VNF such as its functionality, operations, etc. They also include details about the VNF provider, cost, location where the VNF provider saved it. This location end-point is the key for CDN providers since they have to fetch VNF, download and deploy in NFVI premises for offering services to end-users. VNF providers can extend VNF descriptor by including attributes whenever necessary. Execution operation specifies that the transcoding operation will be performed with input as a video file. Management operations can include one or many functionalities. One operation specified is for converting from AVI to mp4 format. Other, maybe from mp4 to AVI format. VNF properties concerning requirements and the capabilities are also added. The image format can be any of the following such as iso, QCOW2, raw, or any. Access specifiers mention whether it is publically accessible or private. VNF characteristics hold details on billing accounting model, cost of the VNF to purchase, the name of the provider, runtime, and execution dependencies if any, etc.

WSDL has four main fields such as type, message, portType, and binding. Type defines the data type used, message defines the data elements for each operation, portType specifies the operations that are performed and binding represents the protocol and data format for each port type. Since

CDN providers are responsible for deploying and offering services for VNFs, we do not use binding fields of WSDL. portType elements define web service, operations performed and messages involved. We make use of portType element to include all the details related to VNF operations. Although request-response is the most common operation type, WSDL also supports one-way operation. This operation basically receives a message but returns no response. We use this type for describing our VNF operations. Message element defines the parts of each message and associated data types for each operation specified. Type element defines the XML schema. We use this element to include all the data specified by VNF providers specific to its functional and non-functional properties. It is possible to use documentation element inside our type element to include more meaningful data and make it more human readable and understandable. Also, for the elements defined inside types, we have included either a default or fixed value. The default value is automatically assigned when no other value is specified. Fixed value is also assigned automatically to attribute and we cannot specify any other values.

In precise, we make use of types, message and portType elements of WSDL to exhaustively describe our VNF. The below Figure 4-1 and Figure 4-2 represents descriptor of transcoder VNF and mixer VNF in WSDL.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<wsdl:definitions xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:vnf="http://www.example.org/Transcoder/"
 xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" xmlns:xsd="http://www.w3.org/2001/XMLSchema" name="Transcoder"
 targetNamespace="http://www.example.org/Transcoder/">
  <wsdl:types>
  <wsdl:documentation>Specifies characteristics of VNF</wsdl:documentation>
    <xsd:schema targetNamespace="http://www.example.org/Transcoder/">
     <xsd:element name="vnf_characteristics">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="vnf_provider" type="xsd:string" fixed="Ericsson"/>
            <xsd:element name="vnf_billing" type="xsd:string" fixed="Type-1"/>
            <xsd:element name="vnf_address" type="xsd:string" fixed="172.17.0.4"/>
            <xsd:element name="vnf_sla_template" type="xsd:string" fixed="Model-1"/>
            <xsd:element name="vnf_pricing" type="xsd:string" fixed="100$"/>
            <xsd:element name="os" type="xsd:string" fixed="linux"/>
            <xsd:element name="architecture" type="xsd:string" fixed="64_bit"/>
            <xsd:element name="execution_environment" type="xsd:string" fixed="jre"/>
            <xsd:element name="runtime_environment" type="xsd:string" fixed="jvm"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="vnf_properties">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="format" type="xsd:string" fixed="QCOW2"/>
            <xsd:element name="vnf_type" type="xsd:string" fixed="server"/>
            <xsd:element name="accesss_specifiers" type="xsd:string" fixed="public"/>
            <xsd:element name="threshold" type="xsd:string" fixed="10 users/instance"/>
            <xsd:element name="size" type="xsd:string" fixed="256 GB"/>
            <xsd:element name="disk_space" type="xsd:string" fixed="2 GB"/>
            <xsd:element name="virtual_CPU" type="xsd:string" fixed="4 CPU"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="TranscodingOperation"></xsd:element>
    </xsd:schema>
  </wsdl:types>
  <wsdl:message name="convert_video_format">
  <wsdl:documentation>Specifies input  output format</wsdl:documentation>
    <wsdl:part element="vnf:TranscodingOperation" name="inputavi"/>
    <wsdl:part element="vnf:TranscodingOperation" name="outputmp4"/>
    <wsdl:part element="vnf:TranscodingOperation" name="inputmp4"/>
    <wsdl:part element="vnf:TranscodingOperation" name="outputavi"/>
  </wsdl:message>
  <wsdl:portType name="Transcoder">
   <wsdl:documentation>Specifies operation</wsdl:documentation>
    <wsdl:operation name="TranscodingOperation">
      <wsdl:input message="vnf:convert_video_format"/>
    </wsdl:operation>
  </wsdl:portType>
</wsdl:definitions>
```

Figure 4-1: Transcoder descriptor in WSDL

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<wsdl:definitions xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:vnf="http://www.vnf.com/Mixer/"
 xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" xmlns:xsd="http://www.w3.org/2001/XMLSchema" name="Mixer"
 targetNamespace="http://www.vnf.com/Mixer.wsdl">
  <wsdl:types>
  <wsdl:documentation>Specifies characteristics of Mixer VNF</wsdl:documentation>
    <xsd:schema targetNamespace="http://www.vnf.com/Mixer.xsd">
     <xsd:element name="vnf_characteristics">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="vnf_provider" type="xsd:string" fixed="Cisco"/>
            <xsd:element name="vnf_billing" type="xsd:string" fixed="Type-3"/>
            <xsd:element name="vnf_address" type="xsd:string" fixed="172.17.0.8"/>
            <xsd:element name="vnf_sla_template" type="xsd:string" fixed="Model-7"/>
            <xsd:element name="vnf_pricing" type="xsd:string" fixed="80$"/>
            <xsd:element name="os" type="xsd:string" fixed="linux"/>
            <xsd:element name="architecture" type="xsd:string" fixed="32_bit"/>
            <xsd:element name="execution_environment" type="xsd:string" fixed="jre"/>
            <xsd:element name="runtime_environment" type="xsd:string" fixed="jvm"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="vnf_properties">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="format" type="xsd:string" fixed="ISO"/>
            <xsd:element name="vnf_type" type="xsd:string" fixed="server"/>
            <xsd:element name="accesss_specifiers" type="xsd:string" fixed="private"/>
            <xsd:element name="threshold" type="xsd:string" fixed="100 users/instance"/>
            <xsd:element name="size" type="xsd:string" fixed="512 GB"/>
            <xsd:element name="disk_space" type="xsd:string" fixed="24 GB"/>
            <xsd:element name="virtual_CPU" type="xsd:string" fixed="8 CPU"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="MixingOperation"></xsd:element>
    </xsd:schema>
  </wsdl:types>
  <wsdl:message name="add_overlay">
  <wsdl:documentation>Specifies input  output format</wsdl:documentation>
    <wsdl:part element="vnf:MixingOperation" name="input_video_1"/>
    <wsdl:part element="vnf:MixingOperation" name="input_video_2"/>
    <wsdl:part element="vnf:MixingOperation" name="output_video_3"/>
  </wsdl:message>
  <wsdl:portType name="Mixer">
   <wsdl:documentation>Specifies operation</wsdl:documentation>
    <wsdl:operation name="MixingOperation">
      <wsdl:input message="vnf:add_overlay"/>
    </wsdl:operation>
  </wsdl:portType>
</wsdl:definitions>
```

Figure 4-2: Mixer descriptor in WSDL

4.2 Overall Architecture

In this section, we first discuss the architectural principles that we follow to design the proposed architecture for a VNF publication and discovery. Then we describe the architectural components.

4.2.1 Architectural Principles

The first architectural principle is to use widely used architectural style such as REST [65]. These are stateless, client-server, uniform interface communication protocol which virtually uses HTTP.

The second architectural principle is to use the existing Domain Name Server for the implementation of VNF publication and discovery.

The third architectural principle is to reuse the pre-existing built-in network capability of DNS [66] for discovery methods.

4.2.2 Architectural Components

The proposed architecture consists of three layers such as publication layer, discovery layer, and repository manager layer, as shown in Figure 4-3. These deal with two key facets: (i) Publication of VNF descriptors, (ii) discovery of VNFs.

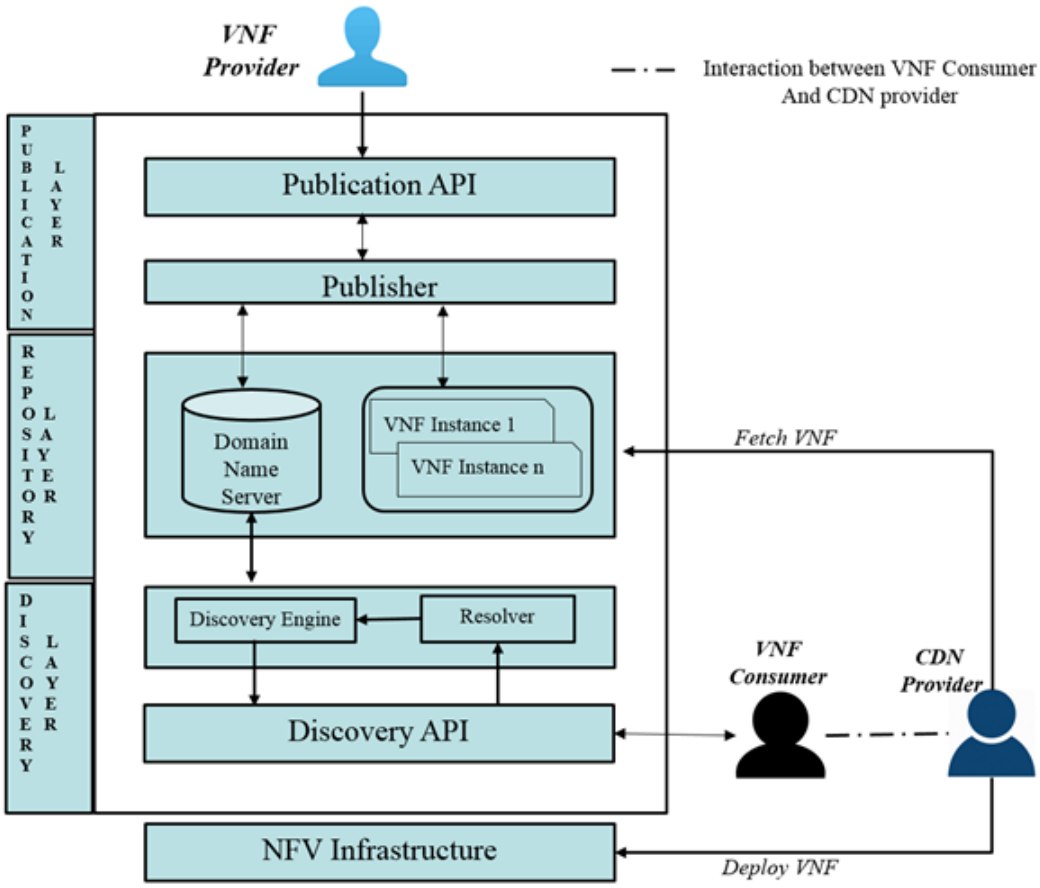

Figure 4-3: Overall Architecture of VNF Publication and Discovery

1) Components related to Publication Layer:

Components related to publication layer in our architecture are *Publication API and Publisher*. VNF developers develop VNFs and create descriptor file for VNFs. *Publication GUI* is responsible for publishing the VNF and WSDL file for VNFs in proper components in repository layer. This uses HTTP in the background. Upon submission, *Publication GUI* uploads VNF descriptor file to the *Publisher* and publish VNF in compute nodes. *Publisher* receives WSDL file of VNF descriptors as an input, reads the entire file and fetches details which are only specific to VNF. These metadata are then modified by the *Publisher* in such a way, to store inside DNS. They are

usually in key-value pair format. *Publisher* then publishes this VNF metadata to the *Repository* layer.

2) Components related to Repository Manager Layer:

Repository manager layer includes a Domain Name Server and compute instances. The main functionality of a Domain Name Server is to translate domain names into IP addresses. This component has zone files associated with domains. Zone files can be distributed among different geographical locations. They are responsible for holding information inside records. Records can be of several types such as A, AAAA, MX, CNAME, NS, TXT, SRV, SOA. Each record includes a specific type of information. We reuse record types such as A, TXT, NS and SOA. A record is used to store the IPv4 address of VNF address published by WSDL file. All the name, fixed values of field types of WSDL file are published inside *TXT* record. This special record can hold the string value and hence can save key-value pair generated, key specifying the name of the VNF and value mentioning the descriptor data. This record is very vital since any service discovery is dependent on the information associated with it.NS record is used to delegate a sub domain to a set of name servers. Value present inside targetNamespace field of WSDL file is parsed and is published inside NS record. SOA record is information stored in a domain name system zone about that zone and other DNS records. This record is used and has several time-specific values. They are set according to our needs and are published.

3) Components related to Discovery Layer:

Discovery layer includes *Discovery GUI, Discovery engine*, and *Resolver* component. Whenever VNF consumer is looking for a VNF, VNF consumer makes use of *Discovery GUI* to discover them. Discovery GUI accepts the VNF request parameters from the VNF consumer. It uses HTTP calls in

the background and upon submission, it forwards the request parameters to *Resolver* component. *Resolver* accepts the input request parameters, create a query based on an input. It then invokes *discovery engine*. *Discovery engine* uses discovery mechanism and uses its CLI interface to make a search on the zone files hosted inside the *Domain Name Server*. Soon after the response received on matching VNF details, it will be sent back to the VNF consumer.

4.3 Illustrative Scenario

The illustrative scenario consists of several actors. The first actor is an end-user who is looking for a video content in his/her device without any interruptions. The second actor is a CDN provider who is responsible for providing transcoding services to end user by associating the content from the content provider. The third actor is a VNF provider who has already developed and offered descriptor for the transcoder. The fourth actor is the VNF consumer who discovers the transcoder VNF upon request from CDN Provider. Repository Manager represents the fifth actor in this scenario.

We divide the scenario into two parts. The first part concerns the publication of transcoder VNF and its descriptors. It illustrates how VNF providers publish VNFs and its descriptors to corresponding components in repository managers. The second part relates to the discovery of transcoder VNFs. It shows how the VNF consumer discovers a VNF when the CDN provider asks for it.

Both parts of the scenario demonstrate the relevant interactions among the different architectural components of the proposed VNF publication and discovery architecture.

4.3.1 VNF Publication

In this scenario, we assume that the VNF provider has developed and published the transcoder VNF in some compute instance. This means he is aware of the location of transcoder VNF and included this detail in transcoder's descriptor already. He now wants to publish a transcoder descriptor, so that stored VNF can be used upon discovery. The transcoder VNF is capable of transcoding videos from one format to another. Utilization of this VNF would help CDN providers to offer end-end services without good Quality of Experience (QoE).

VNF Provider uses *Publication GUI* to upload VNF and descriptor. VNFs will be made available on compute instances and corresponding descriptor files are sent to to the *Publisher*. *Publisher* upon receiving descriptor will read the entire descriptor and fetch the necessary information for further publishing. This uses REST APIs to publish the descriptors to *Domain Name Servers* residing in Repository Manager. *Publisher* can also be used to delete the already residing records in zone files of *Domain Name Servers*. Zone files contain the mappings between domain names and IP addresses. They also include a text representation of *Resource Records*. These text files are more particular to a domain, often a single domain. POST and DELETE methods can be used to add and remove both the forward zone and reverse zone files. The POST method returns 201 after successful creation of a zone file and the DELETE method returns a 200 upon successful deletion. VNF provider enables discovery for VNF consumers soon after the response is received. The VNF provider can use these methods to publish multiple VNF descriptors at the same time. An example for a transcoder VNF's descriptor provided by VNF provider has already shown in Figure 4-1. Figure 4-4 describes an illustrative scenario on VNF Publication of descriptors.

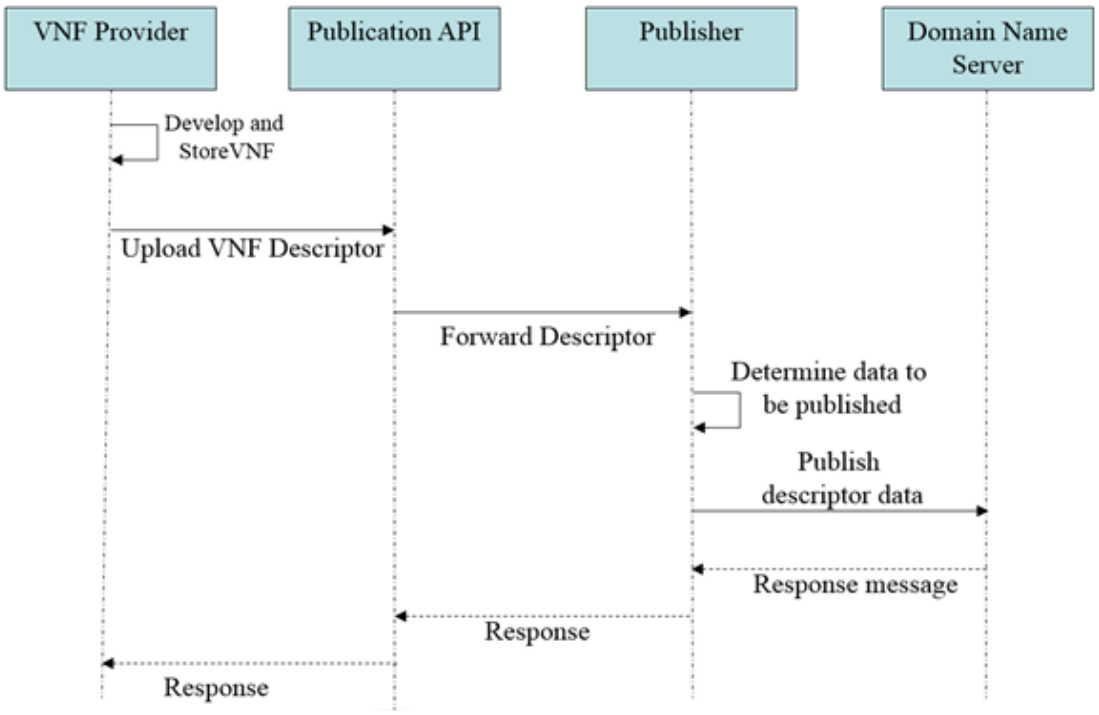

Figure 4-4: VNF Publication Scenario

4.3.2 VNF Discovery

Once the VNF metadata is published, the *Domain Name Server* is ready to handle requests for discovery purposes. Figure 4-5 depicts the interactions when the *resolver* receives a request from VNF consumer for discovering a VNF. End-user is in need of a video in AVI format. When end-user requests the CDN provider, it has to instantiate a VNF for offering transcoding service. Assume, CDN provider has not previously deployed transcoder VNF in its infrastructure and does not know

where to fetch them, satisfying end-user requirements. In this scenario, CDN provider submits his requirements to the VNF consumer and ask him to offer best suitable VNF details along with its location. This will help them in a further download, running and providing the service. VNF consumer, thereby first uses *Discovery GUI* to upload the requirements. *Discovery GUI* invokes the *resolver* to look for transcoder VNF. *Resolver* component decodes the request, fetch necessary information and creates a query. It then sends the query to the to the *Discovery engine*. The *Discovery engine* reads the zone files in the DNS server and when a matching VNF is found, then it responds with all necessary information. DNS also has an inbuilt cache which stores the request parameters coming from *Discovery Engine* and the result. This subsequently reduces future discovery time.

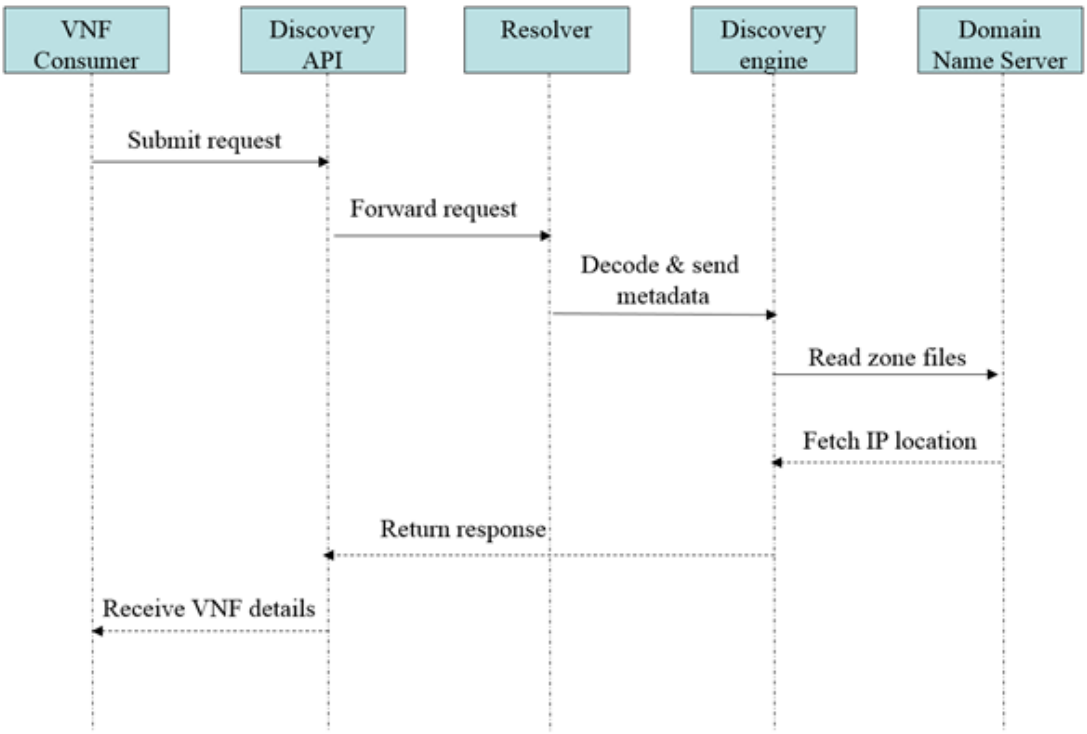

Figure 4-5: VNF Discovery Scenario

4.4 How the Proposed Architecture Meets the Requirements

The proposed architecture of publication and discovery of VNFs satisfies all the requirements mentioned in chapter 3. First, it provides a universal repository for VNF publication and discovery. For publication, VNF provider uses the DNS component inside Repository manager layer to publish descriptors. Also, VNF consumers make use of the same by reading various zone files consisting of resource records and determine IP address for relevant VNF. Second, caching enabled inside DNS enables VNF consumers to determine VNF in less time which in turn helps CDN providers to offer good Quality of Service to end users by offering service with minimized latency. Third, the descriptors offered by VNF providers are geographically distributed in various zones by DNS server which meets our third requirement. Fourth, automatic and reliable discovery procedures are handled by the discovery engine. Finally, the architecture has components for publication and discovery which uses well-defined HTTP and REST interfaces for communicating with them which meets the requirements.

4.5 Chapter Summary

In this chapter, we presented the proposed VNF descriptor model and proposed architecture for publication and discovery of VNFs. We discussed architectural principles that we followed in our design and the main components in our architecture. Next, we provided an illustrative scenario, showing how different components of the proposed architecture communicate with each other and how VNF providers can publish VNF and CDN providers can discover them to offer services to

end-user. Finally, we explained how the proposed architecture fulfills the requirements we set in chapter 3.

Chapter 5

5. Validation: Prototype and Evaluation

In chapter 4, we have proposed a description model for VNFs and a general architecture for a VNF publication and discovery. This chapter focuses on software architecture, prototype implementation and evaluation of the results. First sub-section gives details of the software architecture. Then it is followed by a prototype to validate the software architecture. The last sub-section discusses the performance measurements of the prototype. Lastly, the chapter summary is presented.

5.1 Software Architecture

Figure 5-1 shows the overall software architecture for the proposed publication and discovery mechanism for VNFs. The architecture consists of three layers: the publication layer, the repository layer, and the discovery layer. The publication layer is responsible for publishing VNFs and its descriptors. It consists of Graphical User Interface namely Descriptor GUI and Instance GUI, and components such as parser and publisher. GUI helps VNF provider in uploading VNFs and their descriptors. These components carry out functions which aid in further publication. The Repository layer is responsible to hold the published VNF and its descriptors, published by components in the publication layer. It consists components such as domain name server, caching server, and compute instances running on an Infrastructure as a Service. Furthermore, these components interact with components of Discovery layer. The discovery layer includes a GUI and several components. It has discovery GUI, components such as resolver, discovery engine and forwarder. These are useful in discovering the matching VNF based on requirements. Say an end-user is interested in transcoder

VNF with pricing as 100$ and VNF developed by Ericsson, discovery layer takes responsibility in figuring out the best-matched VNF among all the available ones, and offer them back.

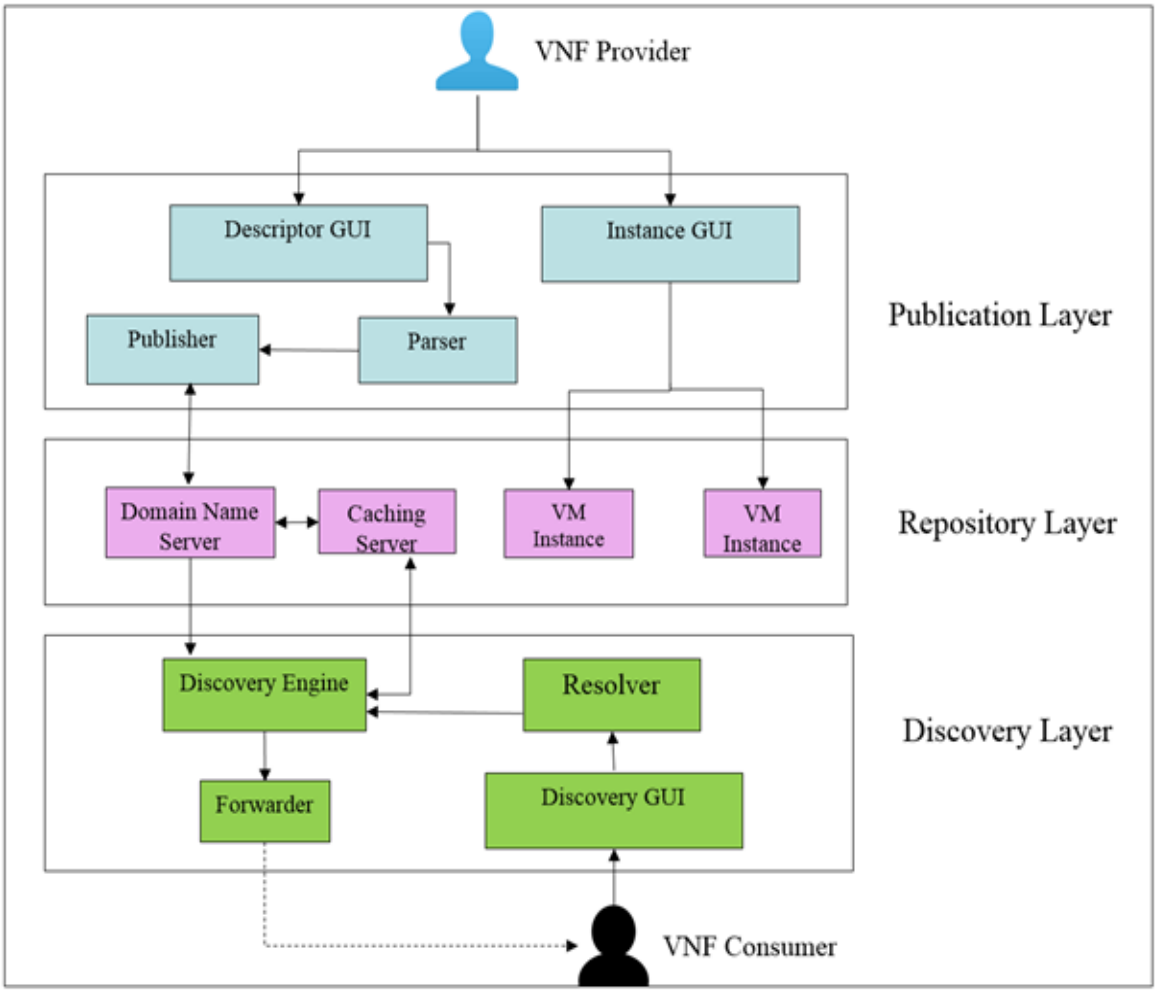

Figure 5-1: Proposed Software Architecture

5.1.1 Operational Procedures

Based on the proposed software architecture, we now present the operational procedures for publication and discovery of VNFs.

Figure 5-2 depicts the workflow of the interaction of software components for publication of VNF and its descriptors. VNF provider uses Instance GUI to publish the VNFs in compute instances. It uses Descriptor GUI to upload the VNF descriptor. Upon submission, it forwards the WSDL defined VNF descriptor to Parser component. The parser reads the WSDL file and extracts the necessary values which are sent to the publisher. The publisher is now responsible to publish the extracted VNF descriptor information inside the DNS. It publishes in record types such as TXT, NS, A, and SOA of DNS zone files. Zone file can be one or many, but each zone file belongs to a single domain, and hence the publisher can publish the available information in resource records of one or multiple zone files. It then receives a return response soon after it is published.

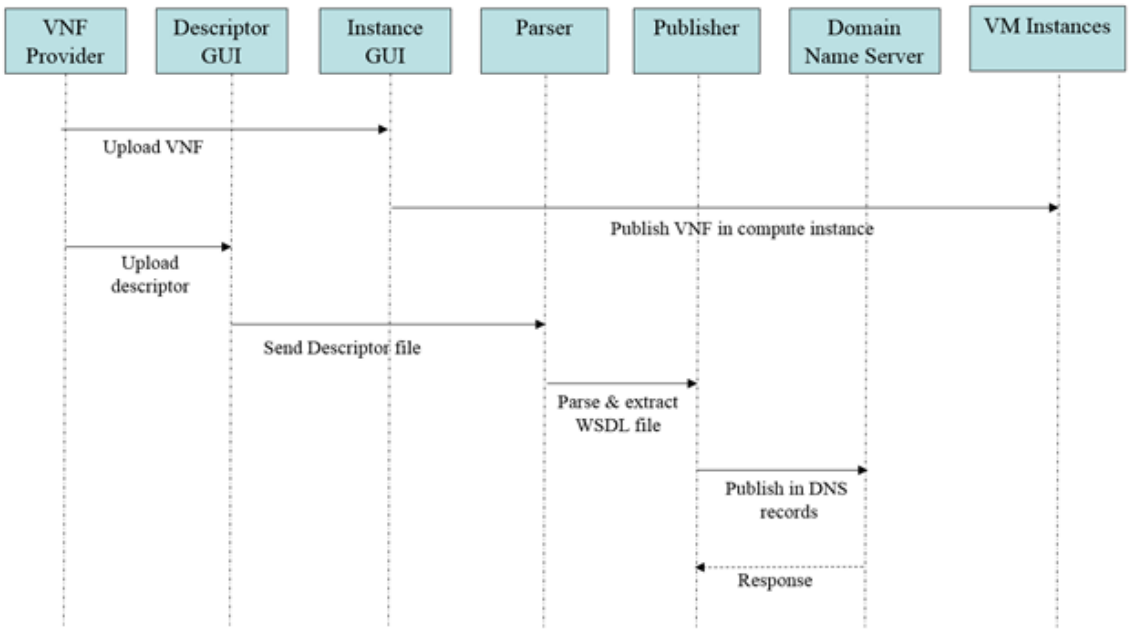

Figure 5-2: Interactions of software components during Publication

Figure 5-3 illustrates the interaction of various software components during the discovery process. VNF consumer uses Discovery GUI to submit the VNF requirements. Upon submission, it sends the request to the resolver component. Resolver components decode the request, extract the parameters and frame query. It then sends the query to the discovery engine. Discovery engine now searches in the cache server. When a cache miss happens, cache server forwards the request to the Domain name server. The response of matching VNF details are written in the forwarder by the discovery engine. VNF consumer can further make use of this for deploying VNFs in its NFVI premises and offering services to end user.

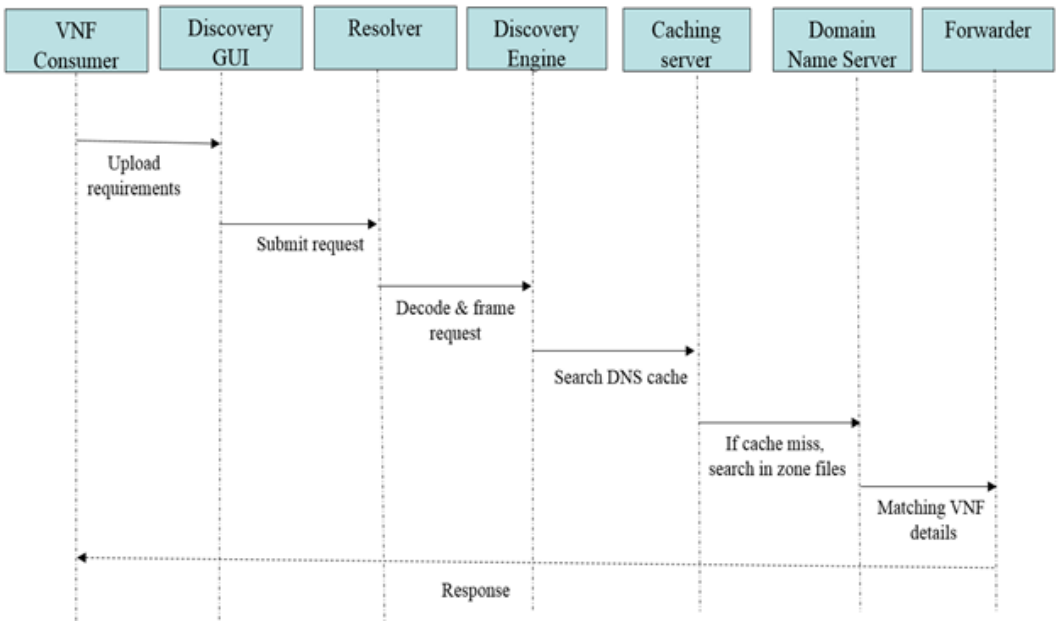

Figure 5-3: Interaction of software components during Discovery

5.2 Prototype

This subsection first presents the implemented scenario. Then we discuss the software tools used for implementation of our prototype. Followed by that, we present the prototype architecture. Lastly, the prototype setup is discussed briefly

5.2.1 Implemented Scenario

We have implemented a value-added service provisioning application inspired by the scenario presented in Chapter 3. In this implementation, we consider that the Virtual Network Functions are already developed by VNF Provider.

To measure the performance of the proposed architecture four scenarios were implemented. Based on these four scenarios, measurements are taken for the proposed architecture. The four scenarios are divided into two groups. Group 1 focuses on caching functionality. Group 2 mainly focuses on the search criterion. We evaluate the performance of our proposed architecture based on different scenarios. Following are the two groups of scenarios:

Group 1:

1 Scenario 1: Caching enabled

In this scenario, we have configured our BIND server with caching capabilities. Since caching is enabled, the discovery engine will first query the cache server. When a cache miss happens, then the request will be forwarded to the domain name server. Also, the result obtained after processing will be saved in caching server by the domain name server, before it is sent back. This scenario offers the reduced latency compared to other.

2 Scenario 2: Caching disabled

In this scenario, we have configured our BIND server without any caching capabilities. Since caching is disabled, the discovery engine will directly query the name server. When results are obtained, it does not cache any. For subsequent requests in future, domain name server treats as if it is received for the first time and start the discovery process by reading the zone files one by one. This scenario offers high latency when compared to the previous scenario.

Group 2:

3 Scenario 3: Single requirement

In this scenario, we modified the system to accept only one requirement submitted by the VNF consumer and further proceed with the discovery process. Although DNS can be queried by multiple clients, we configured the system here to accept only one search criterion by a single or multiple clients. This scenario tests the performance of the discovery mechanism and can be further evaluated upon two sub scenarios-caching enabled and caching disabled.

4 Scenario 4: Multiple requirements

In this scenario, we configured the system to accept multiple search criteria and further query the DNS. Since DNS can be queried by multiple clients, we configured the system to accept multiple search criterion by a single user and multiple users. This scenario tests the performance of the discovery mechanism and can also be verified upon two sub-scenarios- caching enabled and disabled.

5.2.2 Software tools

This section briefly discusses the software tools used in our prototype implementation.

5.2.3.1 Eclipse

Eclipse [56] is an integrated development environment (IDE) used in computer programming, and is the most widely used Java IDE. It contains a base workspace and an extensible plug-in system for customizing the environment. Eclipse is written mostly in Java and its primary use is for developing Java applications, but it may also be used to develop applications in other programming languages.

5.2.3.2 BIND

BIND [57] is open source software that enables you to publish your Domain Name System (DNS) information on the Internet and to resolve DNS queries for your users. The name BIND stands for "Berkeley Internet Name Domain". BIND implements the DNS protocols. The DNS protocols are part of the core Internet standards.

5.2.3.3 OpenStack

OpenStack [58] is a collection of open source software projects that cloud providers can use to setup and run their infrastructure. It works with popular enterprise and open source technologies making it ideal for heterogeneous infrastructure. It has a community of researchers, developers, and enterprises, with a common goal to create simple, scalable and feature-rich infrastructure. OpenStack provides services such as compute, storage and networking resources which can be managed via

dashboard or API. It also has identify service and VM image service. The graphical dashboard helps in managing virtual machines and networks. They also offer a marketplace as part of their commercial support.

5.2.3.4 SAVI

Smart Applications on Virtual Network (SAVI) [59] is collaboration among Canadian industry, research, academia and education networks. Its goal is to investigate key elements of future application platforms. The testbed provides flexible, virtualized infrastructure to support experimental research. This testbed is implemented using OpenStack, which is an Infrastructure as a Service. It implements most of the available features of OpenStack with various geographical nodes.

5.2.3.5 FileZilla

FileZilla [60] is a cross-platform graphical FTP, SFTP, and FTPS file management tool for Windows, Linux, Mac OS X, and more. With tons of intuitive tools, FileZilla helps you quickly move files between your computer and Web server. FileZilla is a reliable, accessible program with many basic functions and advanced tools for expert users

5.2.3.6 NSlookup

Nslookup [61] is a network administration command-line tool available for many computer operating systems for querying the Domain Name System to obtain domain name or IP address mapping or for any other specific DNS record. Depending on the system, the default query may be

to the local DNS name server at your service provider or intermediate name server, or root server system for the entire domain name system hierarchy.

5.2.3.7 Other plugins

DOM Parser [62] is extended to parse WSDL documents. The JavaScript Development Tools (JSDT) [63] provide plug-ins that implement an IDE supporting the development of JavaScript applications and JavaScript within web applications. It adds a JavaScript project type and perspective to the Eclipse Workbench as well as a number of views, editors, wizards, and builders. PyDev [64] is a python plugin for Eclipse, which is an open source. It has various nice features such as code completion, interactive console, debugger.

5.2.3 Prototype Description

Figure 5-4 shows the prototype architecture. Most of the components of the prototype are hosted inside high capacity virtual machines and uses open-source libraries and frameworks. We used JavaScript Development Tools in Eclipse to develop the Descriptor Graphical User Interface. And for Instance GUI, we reused the already available SAVI dashboard. Parser belonging to the publication layer is implemented inside Eclipse by extending the DOM parser to parse WSDL descriptors. Publisher uses PyDev plugin inside Eclipse to publish entries inside the DNS server. For DNS, we use BIND server and enabled caching in the same as part of our implementation. Discovery GUI inside the Discovery Layer uses HTML-Editor plugin in Eclipse to receive the VNF requirements and we implemented the Resolver component in our Eclipse Discovery Engine uses

nslookup tool to query the Domain Name Server and caching server. OpenStack running on SAVI acts as Infrastructure Provider to host the virtual machines.

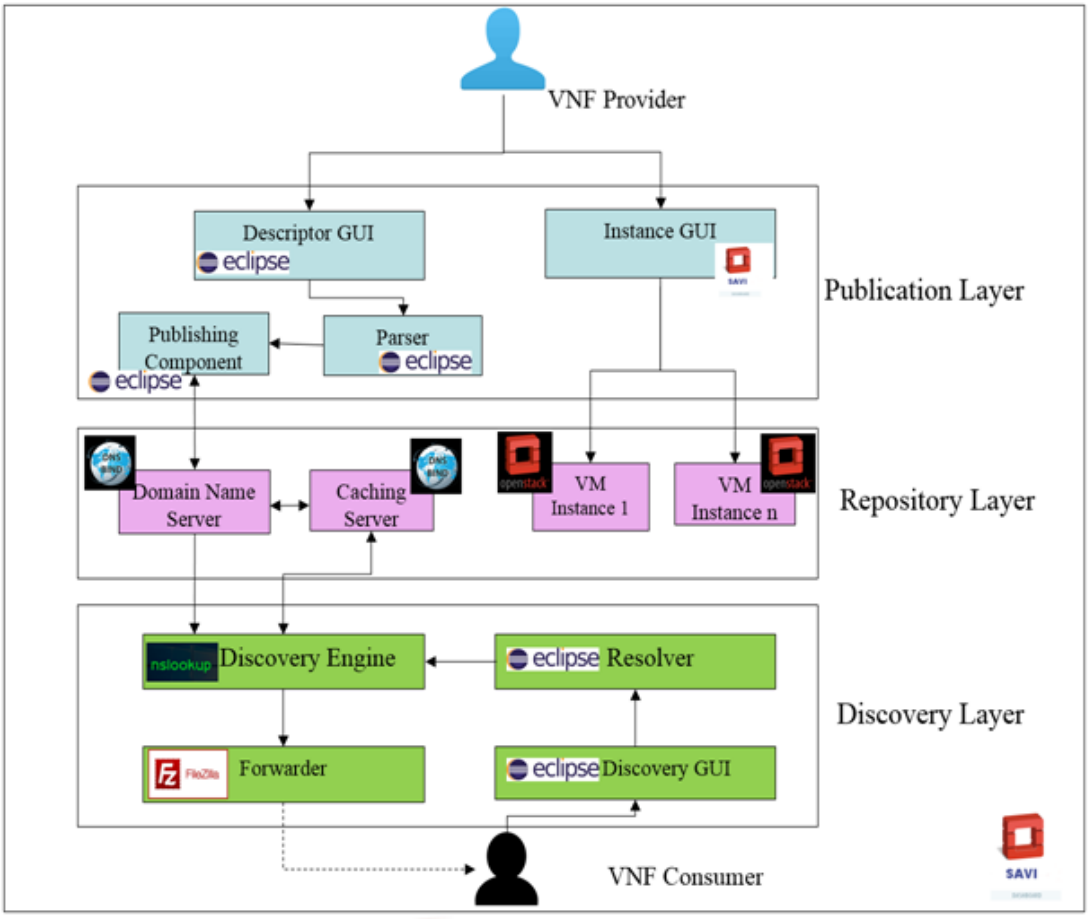

Figure 5-4: Prototype Architecture

5.2.4 Prototype Setup

The entire prototype along with publication, discovery and repository layer components are deployed on SAVI test bed. BIND server is deployed on one VM. It runs on a Linux operating system. All the publication layer components except the Instance GUI runs on one VM. Many VM hosts are used to storing VNFs developed by VNF developers. They are distributed across various geographical

regions. Discovery Layer components are hosted on another VM. Each of these VMs has 8 GB RAM, 4 vCPUs, 80 GB storage and runs Ubuntu 14.04 LTS.

5.3 Performance Measurements

This sub-section discusses various performance metrics. It then discusses the performance results.

5.3.1 Performance metrics

The goal of the performance metrics is to evaluate the time and precision of proposed architecture of VNF discovery and evaluate the performance.

Following are the three metrics are considered:

(i) Round-Trip time: Total time taken by the process to analyze the user requirements, process the query, determine the best match and send back to the client

(ii) Execution time: Time taken by the Discovery engine to search the DNS zone and to compose an ordered list of discovered VNFs

(iii) Recall: Ratio between the number of relevant discovered VNF and the total number of relevant published VNFs.

5.3.2 Performance Results

The results obtained from evaluating the software prototype are described below:

This subsection provides the performance measurement results of the proposed final architecture and analyzed them for each metrics

5.3.2.1 Round-Trip time

Figure 5-5 shows the round-trip time for the scenarios in group 1. Figure 5-6 & Figure 5-7 shows the round-trip time for Scenario 3 & Scenario 4 of Group 2. It is noticeable from the figure that our proposed architecture has an overall round-trip time varying between 4.5-7 seconds depending on various factors. Figure 5-5 shows the average value recorded for different query sets. For this experiment, we experimented with 10 different query sets at random. We can infer that, once the response has been cached, we get the same result of response time when further queried. Figure 5-6 & Figure 5-7 shows the chart for round-trip time calculated by varying the search criterion and by enabling & disabling caching. As inferred by the previous scenario, the graph depicts when a response is cached, round-trip time gets reduced in both scenarios. Also, the time taken to read the zone files is arbitrary but the resolver plays a great role in creating a query and forwarding the same to the discovery engine, which consumes time. For single requirement criterion, round-trip time is obviously lesser than multiple requirement criteria. This is because, creating the query to be forwarded to the discovery engine plays an equally important role, as that of caching.

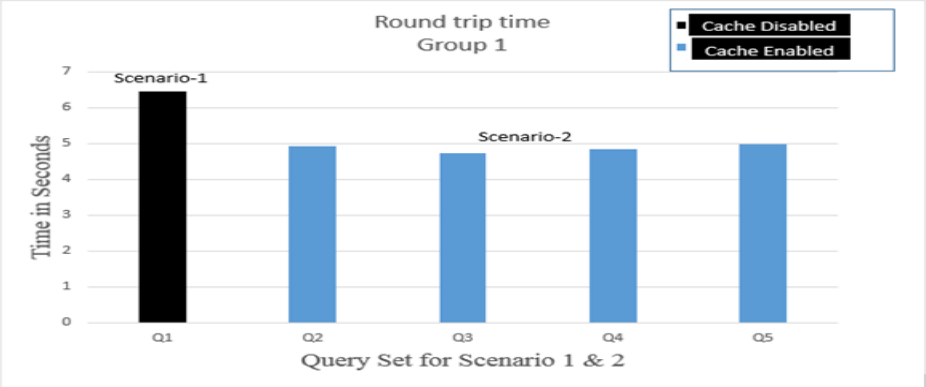

Figure 5-5: Average Round trip time calculated for various Query sets

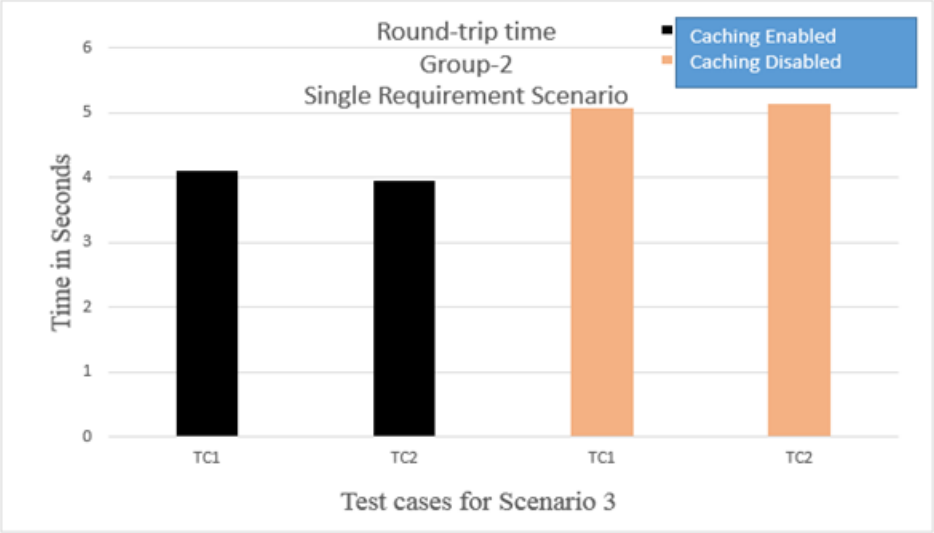

Figure 5-6: Round trip time Group-2 Scenario-3

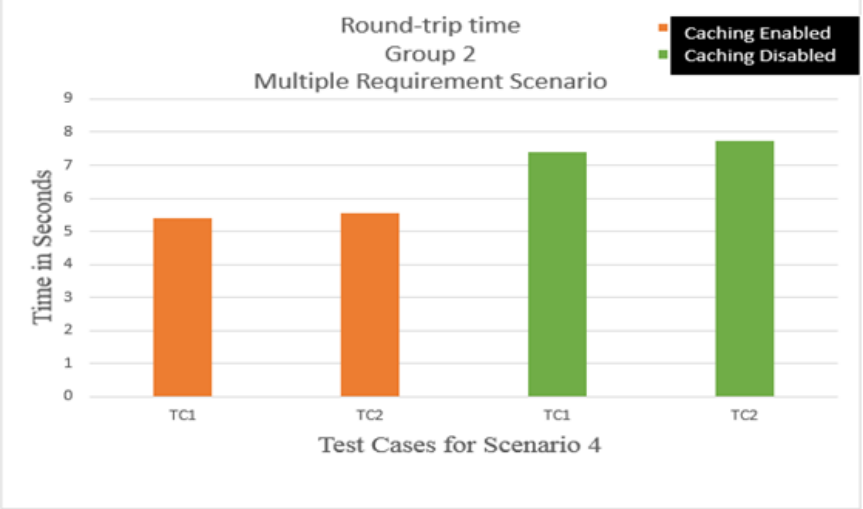

Figure 5-7: Round trip time Group-2 Scenario-4

5.3.2.2 Execution time

Figure 5-8 shows the average execution time calculated for various query sets. Figure 5-9 & Figure 5-10 shows the execution time for scenario 3 & 4 of Group 2 respectively. From Fig 5.8, we can

depict that average execution time of our proposed architecture varies between 2-3.5 seconds. This takes caching into account. Caching server when enabled, our execution scenario takes very less time. When compared with the previous metrics, we can infer execution takes a reasonable amount of time by enabling caching. BIND server which we used is performing optimally even though many records are pushed. From Figure 5-9 & 5-10, we can say that our system shows a vivid time gap only when caching is disabled and not when multiple requirements are sent through the query.

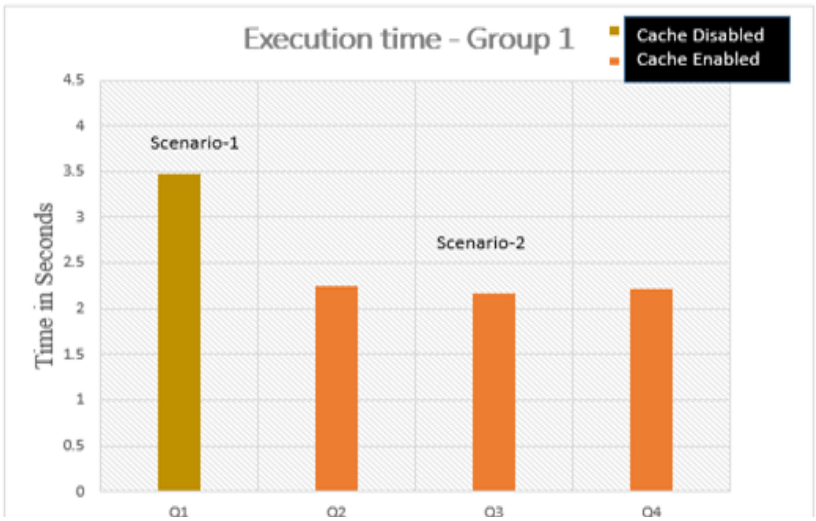

Figure 5-8: Average Execution time calculated for various Query sets

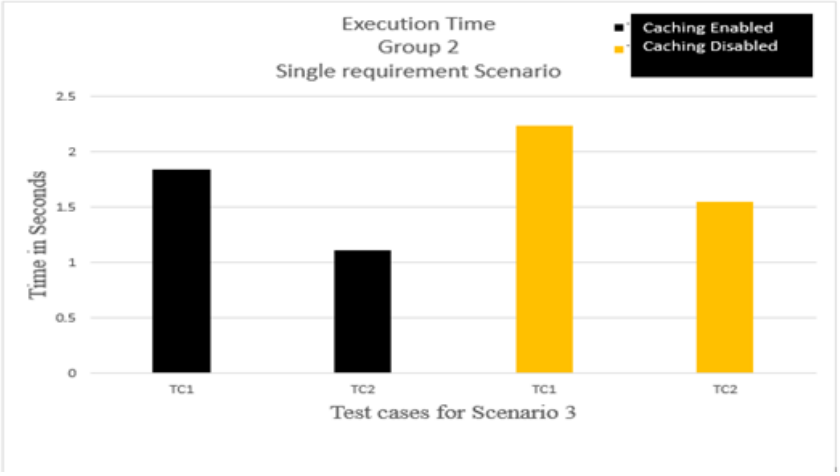

Figure 5-9: Execution time Group-2 Scenario-3

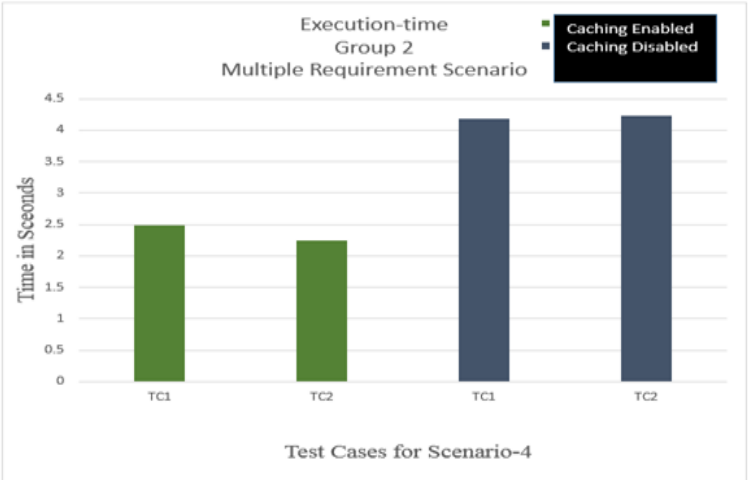

Figure 5-10: Execution time Group-2 Scenario-4

5.3.2.3 Recall

In any of the scenarios above, the relevant VNFs are always discovered by our proposed discovery mechanisms. Thus we verified in our experiments that recall is always equal to one. Recall value

usually ranges between zero to one. A high index of recall means there is little loss of relevant data. Since our recall is one, we can infer that there is no loss of data.

5.3.3 Analysis

From the above analysis, we can infer that our system would work at its best performance when we include caching server close to the end-user. Although caching when enabled in Domain Name Server reduces time, more time is spent on creating the query when multiple requirements are submitted. This consumes a lot of time since most time is spent on forwarding them to the discovery engine. It can be optimized further when we incorporate caching feature. The response received from the Domain Name Server can be cached along with the request parameters, which will subsequently minimize the delay. Also, our proposed software architecture always has a recall of one, which proves that it is the efficient procedure.

5.4 Chapter Summary

In this chapter, we have presented software architecture of VNF Publication and discovery. We have discussed the software components and their interactions for value added service provisioning scenario. We have also explained the prototype and the software tools used to implement the prototype. Then we presented the performance measurements of the prototype with performance metrics and results. In the next chapter, we shall summarize thesis contribution and propose several research directions.

Chapter 6

6. Conclusion and Future Work

In this chapter, we summarize the contributions of this thesis. We also indicate few research directions for future works on VNF publication and discovery.

6.1 Contribution Summary

Virtual network Functions finds its great importance in telecommunication domain. NFV being an agile and fast way of delivering new features into the market, it is extremely vital to offer consumers a choice on VNFs offered by multiple VNF providers. However, this remains a big challenge. One challenge is to describe a VNF in an exhaustive way so that consumers can discover them based on their requirements. Another challenge is efficient publication and discovery procedures. It is critical for VNF providers to publish VNF descriptors and VNF consumers for discovering them to offer efficient services.

We have derived a set of requirements on VNF publication and discovery and evaluated few descriptor models. We have reviewed the state of the art and evaluated them against the requirements. We have identified that none of them meets all of our requirements.

We have proposed a description model for VNFs. This common descriptor model is chosen based on few key principles. We re-used WSDL to describe our VNFs since they are widely accepted standards and provides various capabilities to describe our VNFs. Also, we proposed a novel architecture for publication and discovery of VNFs. Additionally, architectural principles were

identified. Our architecture uses Domain Name Server as a unified repository for publication and discovery procedures. Software architecture and operational principles were also discussed.

A proof of concept prototype is implemented to validate the architecture. The implementation is based on BIND DNS server. The prototype runs on virtual machines which are managed by SAVI. SAVI runs on OpenStack which is an Infrastructure-as-a-service platform. We have evaluated the performance for validating the prototype. From the results, we can infer our proposed architecture has no loss of data, good precision, good execution and response time.

6.2 Future Work

The current architecture consists of an internal cache component. The results show that response time and execution time difference when we enable caching. Also, caching has an effect when there are multiple requirements. It would be nice to include an external cache component close to the end-user, which might even have better response rate. For instance, an end-user looking for a VNF with multiple requirements need not wait until the response is queried and sent back from the DNS server. Instead, by use of external cache component, he would be able to get the response at the minimal time. Another research direction could be to propose an algorithm to select best suitable VNF among various offered VNFs, based on descriptions. Selected VNF could be the one which is already deployed so that consumers can get access to them directly. Such type of approach might have less latency and better performance.

Bibliography

[1] D. Sarkar, N. Rakesh, K. Mishra, “Content Delivery Networks: Insight and Recent Advancements”, 2016 Fourth International Conference on Parallel, Distributed and Grid Computing (PDGC)

[2] C. Makaya, D. Freimuth, D. Wood, S. Calo, "Policy-based NFV management and orchestration”, IEEE Conference on Network Function Virtualization and Software Defined Network (NFVSDN), pp. 128-134, 2015.

[3] J. Xia, Z. Cai, M. Xu, “Optimized Virtual Network Functions Migration for NFV”, 2016 IEEE 22nd International Conference on Parallel and Distributed Systems (ICPADS), 2016

[4] D. Conrad, “A Quick Introduction to the Domain Name System [A]. 2001:Proceedings of the ITU ENUM Workshop[C]”, Geneva, Switzerland, Jan. 17, 2001.

[5] T. Rooney, "The Domain Name System (DNS)," in Introduction to IP Address Management , 1, Wiley-IEEE Press, 2010, pp.69-84

[6] Sajithabanu S., Balasundaram S.R., “ Cloud based Content Delivery Network using Genetic Optimization Algorithm for Storage Cost”, 2016 IEEE International Conference on Advanced Networks and Telecommunications Systems (ANTS), 2016.

[7] Plagemann, T., Goebel, V., Mauthe, A., Mathy, L., Turletti, T., and Urvoy-Keller, G. From content distribution to content networks – issues and challenges. Computer Communications,29(5), pp. 551–562, 2006.

[8] A.-M. K. Pathan, R. Buyya, and A. Vakali, "Content delivery networks: Lecture Notes in Electrical Engineering" pp.6-8: Springer-Verlag, Germany, 2008.

[9] A.-M. K. Pathan, R. Buyya, and A. Vakali, "Content delivery networks: State of the art, insights, and imperatives”, pp. 3-32: Springer-Verlag, Germany, 2008.

[10] G. Pierre and M. van Steen, “Globule: A collaborative content delivery network,” IEEE Communications Magazine, vol. 44, no. 8, pp. 127-133, 2006.

[11] Coral M. J. Freedman, C. Aperjis, and R. Johari, “Prices are right: Managing resources and incentives in peer-assisted content distribution,” Proc. 7th International Workshop on Peer-to-Peer Systems (IPTPS'08), 2008.

[12] L. Wang, K. S. Park, R. Pang, V. Pai, and L. Peterson, “Reliability and security in the CoDeeN content distribution network,” Proc. USENIX 2004 Annual Technical Conference (USENIX'04), USENIX Association Berkeley, CA, USA, 2004.

[13] G. Fortino, C. Mastroianni, and W. Russo, "Collaborative media streaming services based on CDNs," Content Delivery Networks, R. Buyya, A.-M. K.Pathan and A. Vakali, eds., pp. 297-316: Springer-Verlag, Germany, 2008.

[14] J. Sahoo, M. Salahuddin , R. Glitho, H. Elbiaze, W. Ajib, “A Survey on Replica Server Placement Algorithms for Content Delivery Networks”, IEEE Communications Surveys & Tutorials, Vol. 19, No. 2, Second quarter 2017

[15] G. Pallis and A. Vakali, “Insight and Perspectives for Content Delivery Networks,”Comm. ACM, vol. 49, no. 1, ACM Press, 2006, pp.101-102.

[16] nfv white paper etsi. https://portal.etsi.org/NFV/NFV_White_Paper.pdf.

[17] Bo Han, Vijay Gopalakrishnan, Lusheng Ji, and Seungjoon Lee. Network function virtualization: Challenges and opportunities for innovations. IEEE Communications Magazine, 2015.

[18]http://www.etsi.org/deliver/etsi_gs/NFV/001_099/002/01.01.01_60/gs_nfv002v010101p.pdf.

[19] ETSI GS NFV 002 V1.2.1 (2014-12), “Network Functions Virtualization (NFV), Architectural Framework”.

[20] Rashid Mijumbi, Joan Serrat, Juan-Luis Gorricho, Niels Bouten, Filip De Turck, Raouf Boutaba, “Network Function Virtualization: State-of-the-art and Research Challenges”, IEEE Communications Surveys and Tutorials,2015.

[21] ETSI GS NFV 001 V1.1.1 (2013-10), “Network Functions Virtualization (NFV) Use Cases”.

[22] NetFlix, “Fault Tolerance in a High Volume, Distributed System,” http://techblog.netflix.com/2012/02/fault-tolerance-in-high-volume.html , 2012.

[23] Continuity Software, “2014 Service Availability Benchmark Survey,” http://www.continuitysoftware.com/wp-content/uploads/2014/05/2014-SA-Survey-Report.pdf , 2014.

[24] Kurose, James F. and Ross, Keith W. Computer Networking – A Top-Down Approach Featuring the Internet. Addison-Wesley, 2nd Edition, 2003.

[25] P. Mockapetris, "Domain Names - Implementation and Specification", IETF, RFC 1035,1984.

[26] P. Mockapetris, "Domain Names - Concepts and Facilities", IETF, RFC 1034,1984.

[27] N. Alfardan, "On the Design and Implementation of Secure Network Protocols", Ph.D.dissertation, Royal Holloway, University of London, London, UK, 2014.

[28] Kurose, James F. and Ross, Keith W. Computer Networking – A Top-Down Approach Featuring the Internet. Addison-Wesley, 2nd Edition, 2003.

[29] Albitz, P & Liu, C., 2001. DNS and BIND. 4th Ed. Sebastopol, California: O'Reilly Media, Inc.

[30] J. Jung, E. Sit, H. Balakrishnan, and R. Morris, “Dns performance and the effectiveness of caching,” Networking, IEEE/ACM Transactions on,vol. 10, no. 5, pp. 589–603, Oct 2002.

[31] B. Han, V. Gopalakrishnan, L. Ji, S. Lee, "Network function virtualization: Challenges and opportunities for innovations", IEEE Commun. Mag., vol. 53, no. 2, pp. 90-97, Feb. 2015.

[32] P. Veitch, M. J. McGrath, and V. Bayon, “An instrumentation and analytics framework for optimal and robust NFV deployment,” IEEE Commun. Mag., vol. 53, no. 2, pp. 126–133, Feb. 2015.

[33] N. Bouten, J. Famaey, R. Mijumbi, B. Naudts, J. Serrat, S. Latré, and F. D. Turck, “Towards NFV-based multimedia delivery,” in 2015 IFIP/IEEE International Symposium on Integrated Network Management (IM), Ottawa, ON, Canada, pp. 738–741, May 2015.

[34] N. Jahromi, S. Yangui, A. Larabi, D. Smith, M.A. Salahuddin, R. Glitho, R. Brunner, H.Elbiaze, “NFV and SDN-based Cost-efficient and Agile Value-added Video Services Provisioning in Content Delivery Networks”, The 13th Annual IEEE Consumer Communications & Networking Conference, Jan 2016 [35] “Boilerplates.” [Online]. Available: https://www.ng.bluemix.net/docs/starters/boilerplates.html. [Accessed: 05-Nov-2015].

[35]Christian Makaya - Douglas Freimuth , “Automated virtual network functions onboarding” - 2016 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN) – 2016

[36]Evangelos Markakis - Anargyros Sideris - George Alexiou - Athina Bourdena - Evangelos Pallis - George Mastorakis - Constandinos X.Mavromoustakis , “A virtual network functions brokering mechanism” - 2016 International Conference on Telecommunications and Multimedia (TEMU) - 2016

[37]G. Xilouris - E. Trouva - F. Lobillo - J. M.Soares - J. Carapinha - M. J.Mcgrath - G. Gardikis - P. Paglierani - E. Pallis - L. Zuccaro - Y. Rebahi - A. Kourtis, “T-NOVA: A marketplace for virtualized network functions” 2014 European Conference on Networks and Communications (EuCNC) - 2014

[38]OpenBaton: http://openbaton.github.io/

[39]M. AbuJarour and F. Naumann, "Towards a Diamond SOA Operational Model," in SOCA. IEEE, 2010, pp. 1-4.

[40]G.Hong-jie, M Fan-rong, S Jin-fei, “Web Service Discovery Based on the Cooperation of UDDI and DF”, 4th International Conference on Wireless Communications, Networking and Mobile Computing-2008

[41] B. Jin, L. Zhang, Z. Zang, "A Unified Service Discovery Framework", Proc. Sixth Int'l Conf. Grid and Cooperative Computing (GCC '07), pp. 203-209, 2007.

[42] Y. Feng, Q. Li, “The Distributed UDDI System Model Based on Service Oriented Architecture”, 7th IEEE International Conference on Software Engineering and Service Science-2016

[43] M. J. Hadley. Web Application Description Language (WADL). Technical report, Sun Microsystems, November 2006. Available at https://wadl.dev.java.net/

[44] Amazon.com. Amazon Web Services. Technical report, Amazon.com, 2005. See http://www.amazon.com/.

[45] Ran, S.: A Model for Web Services Discovery with QoS. SIGecom Exch. 4(1), 1–10 (2003)

[46] https://github.com/nfvlabs/openmano/wiki/openmano-descriptors

[47] http://openbaton.github.io/documentation/ns-descriptor/]

[48] http://www.etsi.org/deliver/etsi_gs/NFV-MAN/001_099/001/01.01.01_60/gs_NFV-MAN001v010101p.pdf

[49]http://docs.oasis-open.org/tosca/tosca-nfv/v1.0/csd03/tosca-nfv-v1.0-csd03.pdf

[50]https://tools.ietf.org/id/draft-aranda-nfvrg-recursive-vnf-00.html#rfc.section.3.1

[51] F. Curbera, "Unraveling the Web Services Web", IEEE Internet Computing, vol. 6, no. 2, pp. 86-93, 2002.

[52] Marc J. Hadley, Web application description language (WADL), Sun Microsystems, Inc., Mountain View, CA, 2006

[53] Ben-Kiki, O., Evans, C., & Ingerson, B. (2009). YAML ain't markup language (YAML) version 1.2. http://www.yaml.org/spec/1.2/spec.html.

[54] https://www.w3.org/Submission/wadl/

[55] J. van Sloten, A. Pras, M. van Sinderen, "On the standardisation of Web service management operations", Proceedings of 10th Open European Summer School School and IFIP WG 6.3 Workshop (EUNICE 2004), pp. 143-150, 2004-June

[56] "Eclipse." [Online]. Available: http://www.eclipse.org/ [Accessed: 11-Sep-2017]

[57] "BIND." [Online]. Available: https://www.isc.org/downloads/bind/ [Accessed: 11-Sep-2017]

[58] [Online]. Available: https://www.openstack.org/ [Accessed: 11-Sep-2017]

[59] [Online]. Available: "Smart Applications on Virtual Infrastructure." [Online]. Available: http://www.savinetwork.ca / [Accessed: 11-Sep-2017]

[60] [Online]. Available: https://filezilla-project.org/ [Accessed: 11-Sep-2017]

[61] [Online]. Available: https://technet.microsoft.com/en-us/library [Accessed: 11-Sep-2017]

[62][Online].Available:https://developer.mozilla.org/en-US/docs/Web/API/DOMParser [Accessed: 11-Sep-2017]

[63] [Online]. Available: https://eclipse.org/webtools/jsdt/ [Accessed: 11-Sep-2017]

[64] [Online]. Available: http://www.pydev.org/index.html [Accessed: 11-Sep-2017]

[65] M. Lanthaler and C. Gutl, "Towards a restful service ecosystem," in Digital Ecosystems and Technologies (DEST), 2010 4th IEEE International Conference on. IEEE, April 2010, pp. 209-214.

[66] Michael Dooley; Timothy Rooney, "Introduction to the Domain Name System (DNS)," in DNS Security Management , 1, Wiley-IEEE Press, 2017, pp.324-doi: 10.1002/9781119328292.ch2