vConductor: An NFV Management Solution for Realizing End-to-End Virtual Network Services

Wenyu Shen, Masahiro Yoshida, Taichi Kawabata, Kenji Minato, Wataru Imajuku
NTT Network Innovation Laboratories
1-1 Hikari-no-oka, Yokosuka, Kanagawa 239-0847 Japan
wenyu.shen@lab.ntt.co.jp

Abstract—NFV, being discussed and standardized in ETSI, is regarded as a promising candidate technology for Future Networks. Although many advantages are expected, the introduction of NFV will greatly challenge current management systems. In this paper, we summarize some of the carrier’s requirements for the management of NFV, based on which we propose vConductor as an NFV management solution. We say vConductor is innovative due to its characteristics of automatic service provisioning, end-to-end inventory management, and generic system design. To achieve these goals, we design a completely modular-based system structure and refine the process of provisioning a virtual network service. Furthermore, we propose a general data model, specifically for NFV, starting from the TM Forum SID. We believe that vConductor will be a catalyst for accelerating the industrialization of NFV.

Keywords—Network Function Virtualization; Orchestrator; Data Modelling; Network Service Chaining

I. INTRODUCTION

Network functions virtualization (NFV), originally proposed in the European Telecommunications Standards Institute (ETSI), has recently become a world-wide hot topic [1]. Both academia and industry have come to regard it as a promising candidate for the so-called Future Networks, along with Software Defined Networking [1, 2]. According to the NFV concept, network functions such as firewalls, routers, network address translations (NATs), which used to be realized as dedicated physical devices, will be completely or partially implemented as software; they are virtualized and installed on universal servers. In fact, a more challenging solution is to deploy these functions to a cloud environment.

A variety of benefits are expected with the introduction of NFV [1]. For example, by eliminating dedicated physical devices and fully utilizing the cloud infrastructure, carriers foresee the possibility of significant cost reductions for introducing new services. In addition, software technologies increase network flexibility, making the rapid introduction of new services and functions possible. However, everything good has a cost. The management of a network composed of virtualized network functions (VNFs) appears to be a tougher challenge than the traditional network. At first, completely software-based network construction may bring unexpected complexity, for example, handling a variety of software appliances and logical tunnels. Next, the essence of virtualization requires that the number of managed objects will increase dramatically [3, 4]. In an NFV environment, we manage not only the physical layer, but also the virtualized layer, as well as the mapping between them. The situation may become even more complex when we consider a practical environment within a carrier, in which rapid provisioning of end-to-end services is required. Last but not least, other practical issues such as backward compatibility and open innovation should be taken into consideration as well.

Our solution is vConductor, an NFV management solution for implementing end-to-end virtual network services. To the best of the authors’ knowledge, vConductor is the first solution that fully conforms to the NFV end-to-end architecture [5] and the latest NFV management & orchestration (MANO) specifications [6] that are being standardized in ETSI. We list some characteristics of vConductor below.

- vConductor fully automatizes the provisioning of virtual network services. The advantages of vConductor, which include simple graphical user interfaces (GUIs), allow the operator to realize easy and rapid service provisioning.
- vConductor provides a complete solution to end-to-end NFV inventory management. By using vConductor, an operator can easily grasp the inventory changes in both the physical layer and the virtualized layer, spanning customer premises and multiple datacenters.
- vConductor is designed to be highly generic, as backward compatibility and open innovation are one of its important characteristics. In fact, it supports the management of existing physical network functions (PNFs) and VNFs. Moreover, its internal components can be easily replaced by existing or new third-party products.

We achieve these benefits by casting the NFV end-to-end architecture into a practicable system design; we adopt a completely modular-based system structure and refine the provisioning process for a virtual network service. In addition, we extend the Information Framework (SID) model of TM Forum in order to adapt SID to an NFV environment [7].
II. A CARRIER’S PERSPECTIVE ON THE CHALLENGES IN THE MANAGEMENT OF NFV

As we mentioned in the Introduction, although many advantages are expected with the introduction of NFV, there are still great challenges in its management. This section discusses these challenges in detail, especially from a carrier’s perspective.

- Challenge 1: Automatic service provisioning

Speaking of network construction may remind people of those routine tasks such as installation of network devices, laying cables, etc. In the context of NFV, the network devices become VNFs which are implemented as software appliances (e.g. Vyatta [8], Open vSwitch [9]) running on virtual machines (VMs) and the cables become virtualized links (VLs) which are implemented as logical tunnels (e.g. VXLAN, GRE). As the result, the network construction process turns out to be execution of a series of software-based commands and scripts, which is actually quite different from the traditional hardware-based approach; here extra considerations shall be taken on such as the amount of resources that must be reserved to guarantee VNF performance, the selection of a physical container (e.g. a datacenter, a server) for deploying a VNF, and the technology for implementing a VL. When we address the issue from the perspective of a real carrier, the task may be overwhelming. First, it is possible that a large number of VNFs and VLs need to be handled in order to construct a practical network and what’s more, these VNFs and VLs may differ in their specifications such as the resource requirements, deployment policies and implementation technologies. Furthermore, service provisioning is always a carrier’s first priority. Here, in order to activate a service, an operator has to further configure the network such as setting a flow table within a switch or the firewall policies within a firewall; these are also deemed to be a heavy burden.

Therefore, in order to realize the rapid introduction of new services, a fundamental goal of NFV, one of the challenges is to release the operators from tedious and error-prone tasks, in other words, to achieve automatic service provisioning. In fact, the software-based nature of NFV makes automation possible. We see the ultimate goal of the automatic service provisioning as being the situation in which the operator only cares about the business aspect of a service (e.g. chaining of a network service [9]) while he leaves all the provisioning details including the network construction and service activation to the management system.

- Challenge 2: End-to-end inventory management

The network used to be composed of physical entities such as actual network devices, so the management was comparatively simple. However, with the introduction of NFV, a network can now be composed of virtualized entities, which raises the issue of managed objects that can be dynamically constructed or modified; they are no longer static [3, 4]. What makes thing more complicated is that in most cases, the services tend to be end-to-end, which is assumed to span multiple domains (e.g. customer premises, datacenters). Suppose a network service chain containing two VNFs scattered among different datacenters. In this case, the VL that connects the two VNFs will be implemented by the two datacenter networks and the inter-datacenter wide area network (WAN). Furthermore, VNFs can be migrated on the fly according to a user’s requirement or real-time network condition, which triggers reestablishment of the related VLs.

For a carrier, this change actually complicates its inventory management, especially for end-to-end services; in order for a carrier to convince potential customers of the reliability of its NFV service, it is necessary to complete a network view that comprises all the information about the physical layer, the virtualized layer, and the mapping between the two, in other words, to achieve end-to-end inventory management. A shortfall in any of the information can lead to a failed fault management and finally a broken service level agreement (SLA). The ultimate goal of the end-to-end inventory management is to allow the operator to locate the root cause of a service malfunction inside the network efficiently, while determining the impact on potential users when a fault occurs inside the network.

- Challenge 3: Backward compatibility and open innovation considerations

Last but not least, backward compatibility and open innovation are regarded as crucial factors for a carrier to introduce any new technology. Hence, a general and extensible management framework with modularized design and a standardized data model is strongly desired. In the context of NFV, it is not practical for a carrier to substitute all its PNFs with VNFs overnight. Similarly, a variety of new network protocols will be created to facilitate network virtualization; a recent example is to utilize OpenFlow [2] to facilitate the establishment of VLs.

Since NFV is still a new concept, there hasn’t been much related work yet based on our knowledge. Although OpenStack [11] can be a valuable reference, it was originally designed as a datacenter manager and great effort is still needed to make it a real carrier-grade solution. Proof of concept (PoC) demonstrations are actively being conducted in ETSI [12], which have brought us a lot of interesting ideas. However, each demonstration focuses only on one specific topic, so we are still lacking a mature solution that covers the entirety of NFV management.

III. DESIGN OF VCONECTOR

We propose vConductor, an NFV management solution for implementing end-to-end virtual network services. As far we know, vConductor is the first solution to fully conform to the NFV end-to-end architecture [5] and the latest NFV MANO specifications [6]. This section describes the design details of vConductor including the topics of system design, operating sequence and data modeling.

A. System design

Fig. 1 shows the system structure of vConductor. Our design fully conforms to the NFV MANO architecture, which is basically composed of an orchestrator, one or more VNF managers and virtualized infrastructure managers (VIMs) [6]. In Fig. 1, the functions corresponding to the orchestrator are
presented in blue, while those corresponding to the VNF managers are presented in pink; they are regarded as the core of vConductor. In order to facilitate service configuration and monitoring, we further embed a user portal and several element managers (shown in grey) which are implemented as plug-ins. Here, the user portal actually fits into the scope of operations support system (OSS), while the element manager fits into the element management system (EMS) as defined in ETSI [5]. vConductor relies on existing VIMs such as OpenStack [11] for managing datacenters and OpenDayLight [13] for managing WANs, so they are assumed to be external systems (shown in green) connected to vConductor. In the following, we explain each component in detail.

- **Service lifecycle management** controls and coordinates the operations of the internal components and external systems, based on the received service orders regarding service generation, modification and deletion. A service order is first input into a service chain analyzer, where its format and feasibility are validated. We omit the design details of the service order due to space limits; it actually consists of a series of descriptors such as a VNF instance descriptor (VNFD) as discussed in the MANO specification [6]. After validation, the analyzer further calculates the necessary resources (e.g. computer resources and network resources) for realizing the requested service by referring to the various kinds of catalog information stored in the catalog database (DB). However, it is the resource reservation component that actually performs the resource reservation. Regarding it as a complicated process which covers many internal components and external systems, we will discuss it later through a sequence diagram for service provisioning.

- **Resource design** consists of a resource scheduler and a parameter generator. The resource scheduler is designed to decide how virtual network elements including VNFs and VLs are allocated to the NFV’s underlying infrastructures (e.g. datacenters). Our resource scheduling algorithm takes into account a variety of constraints such as utilization charges of datacenters, delays between datacenters, and different operational policies inside a carrier [14, 15]. Besides, in the context of NFV, different stakeholders (e.g. service users, operators, and datacenter owners) probably coexist, so to coordinate the conflict among their interests is another important factor when vConductor schedules the deployment of a service [14]. Based on the result of the service chain analysis mentioned above, the parameter generator generates the necessary logical resources such as IP addresses, tunnel identifications (IDs) for practical resource reservation. The necessary physical/logical resource information necessary for the execution of the resource scheduler and the parameter generator can be retrieved from the resource DB.

- **SLA assurance** is designed to guarantee the SLA of a virtual network service by handling hardware faults and performance degradation occurring in the NFV environment. The idea is simple. To achieve this, it simply duplicates an affected VM and reallocates it. To be more specific, the performance/fault analyzer analyses both the status information of VNFs and their supporting resources, which are retrieved from the VNF managers and the VIMs respectively. In this way, the root cause can be located, based on which the scaling controller further decides if it is necessary to perform resource reallocation and moreover, the most proper performance timing. These details will be discussed in a future paper.

- **Catalog DB** manages both service catalog and element catalog in a form of descriptors such as VNF descriptor (VNFD) [6]. The element catalog stores all the resource information for implementing the elements such as VNFs, VLs and VNF forwarding graphs (VNFFGs) [6]. We connect VNFs with VLs and further configure the VNFFGs, so that a service chain or one part of a service chain can be constructed. Frequently used service chains are summarized as templates and managed as service catalog entries.

- **Instance DB** manages the real-time state of instances such as VNFs, VLs, VNFFGs and services that are generated based on the catalog information. Take VNFs as an example; detailed parameters such as assigned IP addresses, performance indicators and log data are covered, but they change throughout the lifecycle of a service. In addition, the instance DB, regarded as a key enabler for the said end-to-end inventory management, also manages the large amount of correlations among the elements of a service such as the combination of VNFs, VLs and VNFFGs to form a virtual service, the allocation of a VNF to a datacenter, the connection of several logical tunnels to form a VL, the implementation details of which will be presented in the following data modeling discussion.

- **Resource DB** manages both the physical resources and logical resources and interacts with VIMs to update the information on a regular basis.

- **Embedded VNF manager** manages the lifecycles of its VNFs, which covers (un)installation of virtual deployment units (VDUs) of a VNF and their initialization and termination [6]. The VNF manager also calls element managers for the initial configuration of a VNF. In order to realize the above management, the VNF manager actually holds a manifest file for each VNF package. Considering future extension, we
design VNF managers in a form of plug-ins, since we note that they are developed by the same developer as VNFs and different from VNF to VNF.

- **User portal** provides an editor for end users to customize their own virtual network services. Our design principle is to minimize the user’s efforts to construct a customized service, so a key goal is a GUI where to construct a service, the only thing that a user needs to do is to drag and drop some icons that represent VNFs, PNFs, VLs, and connect them together. The user inputs will be translated into a special format of service order (see our explanation in the service lifecycle management) before it can finally communicate with the orchestrator. In addition, a service-level monitoring service is also provided, so a user can always grasp the real-time condition of his services.

- **Element managers** are designed as plug-ins since they strongly depend on the corresponding VNFs and PNFs and can be replaced. Basically, they are in charge of the configuration and monitoring of individual elements.

### B. Operating sequence

Based on the system structure above, we further refine the provisioning process of a virtual network service, the result of which is illustrated in Fig. 2. The clarification of the operating sequence not only helps us understand the functioning of individual components inside vConductor, but also enables said automatic service provisioning. Please note that Fig. 2 only covers the initial establishment of a service, while other issues such as the process of auto-scaling are regarded as future work. We explain the provisioning process below.

- **Step 1**: The service lifecycle management receives service orders such as VNFIDs and VLIDs from the user portal [6].
- **Step 2**: The service lifecycle management polls and retrieves the necessary resource-related information such as VNFD and VLD from the catalog DB [6].
- **Step 3**: Based on the received service orders and resource-related information, the service lifecycle management further calculates the resource required for running the requested service and sends it to the resource design function.
- **Step 4**: The resource design function polls the resource DB for available physical resources (e.g., available CPUs, memories) and decides the deployment destination for individual VNFs.
- **Step 5**: Similarly, the resource design function polls the resource DB for the available logical resources (e.g., available IP addresses, tunnel IDs) and designs the configuration parameters for implementing VNFs and VLs.
- **Step 6**: Based on the design results, the service lifecycle management establishes the inter-datacenter tunnels, interacting with the WAN manager.
- **Step 7**: The service lifecycle management further reserves the computer resources and establishes the inner-datacenter tunnels, interacting with the datacenter manager.
- **Step 8**: After the completion of reservation, the VNF manager installs and initializes the component VDUs and performs the VNF initial configuration including the configuration of VNFFG.
- **Step 9**: The instances related to the deployed service including VNFs, VLs, and VNFFGs are registered to the instance DB.

### C. Data Modeling

In order to realize end-to-end inventory management, and generality, we further propose a powerful data model that originates from the SID model [7]. First of all, since our data model completely conforms to the SID model, it is regarded as being general, able to handle networks with multi-vendors and multi-technologies, and to have high affinity with many existing OSSs and EMSs. We further extend SID so that it can support newly proposed NFV concepts such as VNFs, VLs and VNFFGs. As another feature, our data model is able to manage a variety of resource correlations in both the computer domain and the network domain, especially for the end-to-end environment. Design details are shown in Fig. 3, 4, and 5, in which the classes and relationships shown in yellow represent the content that has been already standardized in TM Forum, while those in red are the extended parts, our contributions.
To begin with, we discuss service-related modeling. In Fig. 3, NetworkService represents a virtual network service and is created as a subclass of ResourceFacingService. The associations such as NSRequiresVNFFGs, NSRequiresVLs, NSRequiresVNFs, and NSRequiresPNFs further represent the correlation between a service and its components.

Fig. 4 illustrates IT-resource-related modeling. Here, VNF, PNF, and VDU represent the node elements of a service and are designed as subclasses of LogicalDevice. According to the latest discussion in ETSI [6], a VNF can be further composed of several VDUs, which can be represented by the association of HasVDUs. We design a class named VirtualMachine to represent VMs, so that the concept of server virtualization, the fundamental enabler of NFV, can be illustrated as a series of associations such as OSInstalledOnVM, VDURunsOnOS, VDURealizedBySoftware, and VDUHostedByVM; these enable the management of correlations among a VDU, a program that implements the VDU, the operating system (OS) that supports the execution of the VDU, and finally the container VM. In addition, referring to the modeling of network domains in the exiting SID model [7], we further utilize the composite/atomic pattern to model computer domains for NFV; we design a new Computer class on inheritance from ResourceCollection. We use the ComputerComposite class and ComputerAtomic class to represent the existence of multiple datacenters (Here we regard Datacenter as a subclass of ComputerAtomic). Finally, the class named ResourceServer is introduced so that we can combine a VM and a VNF with the corresponding datacenter that provides the necessary server resources.

- **Network-resource-related modeling**

Fig. 5 illustrates the network-resource-related modeling, in which VL and VNFForwardingGraph, representing the network-related components of a service, are designed as subclasses of PhysicalResource. The associations such as NSRequiresVNFFGs, NSRequiresVLs, NSRequiresVNFs, and NSRequiresPNFs further represent the correlation between a service and its components.

**IV. Prototype**

We are planning a new NFV service which is enabled by vConductor; the service is aimed at integrating cloud services with traditional WAN services. By using this new service, virtual enterprise networks can be created on the fly, even if they span a user’s local premises and multiple remote datacenters. Moreover, based on the design, all the provisioning, including the distribution of software appliances and configuration of logical tunnels, will be performed automatically. Management-as-a-Service (Maas) is regarded as another key characteristic of the service; the operator is responsible for not only the underlying infrastructure, but also the provision of management services to individual users. This can expand the carriers’ business scope to cover enterprise networks.

A PoC demonstration is planned in a future event, however,
in order to show the effectiveness and application of vConductor, this section provides a brief introduction. Fig. 6 illustrates a demonstration GUI image. First, all virtualized elements including VNFs and VLs are presented in the form of icons. As an example, the left part of Fig. 6 presents a list of available VNFs. We can see more details such as the resource catalogs by double clicking an icon. The right part presents a GUI editor that allows the user to customize his network. The shown service chain is only an example, but it covers some software network appliances and server applications. In practice, the service chain can be created and changed by simply dragging and dropping the icons onto arbitrary datacenters (we assume the existence of two datacenters in this example). Optionally, parameters such as IP addresses can also be designated by the user during the customization. Last but not least, we also provide some interfaces for editing the element catalogs and monitoring the individual virtual services (Please see our future PoC for more details). Fig. 7 illustrates a demonstration GUI image. First, all virtualized elements including VNFs and VLs are presented in the form of icons. As an example, the left part of Fig. 6 presents a list of available VNFs. We can see more details such as the resource catalogs by double clicking an icon. The right part presents a GUI editor that allows the user to customize his network. The shown service chain is only an example, but it covers some software network appliances and server applications. In practice, the service chain can be created and changed by simply dragging and dropping the icons onto arbitrary datacenters (we assume the existence of two datacenters in this example). Optionally, parameters such as IP addresses can also be designated by the user during the customization. Last but not least, we also provide some interfaces for editing the element catalogs and monitoring the individual virtual services (Please see our future PoC for more details).

V. CONCLUSION

This paper proposed vConductor, an NFV management solution for conducting end-to-end virtual network services. Our efforts on the system design, provision process refinement and data modeling ensure that vConductor is an innovative solution for realizing automatic service provisioning, end-to-end inventory management, and generality in the context of NFV. As future work on interconnectivity, we plan to discuss the related data model and interfaces with different vendors and carriers to promote standardization. In addition, a PoC demonstration of a new NFV service that is enabled by vConductor will be performed in a future event. Although our research is still in an early stage, we believe that this contribution will be a catalyst for accelerating the industrialization of NFV.

REFERENCE