VNF Service Chaining on SAVI SDI

Pouya Yasrebi[1,2(✉)], Spandan Bemby[1,2], Hadi Bannazadeh[1,2], and Alberto Leon-Garcia[1,2]

[1] University of Toronto, Toronto, ON, Canada
{pouya.yasrebi,spandan.bemby,hadi.bannazadeh, alberto.leongarcia}@utoronto.ca
[2] University of Toronto - St. George Campus, 40 St George St, Toronto, ON M5S 2E4, Canada

Abstract. Managing computational resources and networking elements over today's heterogeneous infrastructure has become very challenging. A need for virtualizing network functions has emerged to reduce infrastructure operating costs. In this paper we consider using software-defined infrastructure (SDI) resource management system (RMS) to achieve service chaining of virtualized network functions (VNFs). SDI allows for the integrated control and management of heterogenous resources. In an SDI environment, the end user has access to interfaces that allow programmatic management of the resources. The user can define their own service graph (SG), which determines the path that traffic must take through various VNFs. The ability to dynamically realize the SG is what is referred to by service chaining. Use cases of service chaining include adding a firewall in front of web server and multicasting. Furthermore, we tested the firewall use case in two scenarios to verify validity of our service chaining implementation.

Keywords: Software defined infrastructure · Network function virtualization · Virtualized network function · Software defined networking · OpenFlow · Smart application on virtual infrastructure

1 Introduction

Application infrastructures consist of two primary components: computing and networking. Due to recent advances in computing and networking technologies, these infrastructures are experiencing two notable changes. First, computing is shifting towards the cloud because of reduced costs through shared resources. Second, traditional networking is being replaced by software-defined networking (SDN) due to its greater flexibility and reduced management cost. Typically, compute and networking resources were controlled and managed separately. However, in [2], Kang et al. presented the software-defined infrastructure (SDI) architecture for a resource management system (RMS) that allows for integrated management of networking and compute resources.

The SDI [6] RMS manages resources in a hierarchical fashion, whereby individual resources are controlled by the corresponding resource controller.

V. Atanasovski, L.-G. Alberto (Eds.): Fabulous 2015, LNICST 159, pp. 11–17, 2015.
DOI: 10.1007/978-3-319-27072-2_2

These resource controllers are managed by a centralized entity, an SDI manager, which allows for integrated management of heterogeneous resources, such as networking, computing, and other unconventional resources like FPGAs and wireless access. The SDI manager also exposes interfaces that allow applications to programmatically manage their resources while having access to topology information and monitoring data. These SDI interfaces also facilitate functions such as orchestration and service chaining.

The SDI RMS is responsible for coordinating all interactions with resources, including the deployment of applications. Deployment consists of various parts including, resource allocation, configuration, and satisfying quality of service (QoS) SLAs. The SDI RMS delegates the allocation and configuration of resources to an orchestration system built upon OpenStack. Orchestration refers to the management, i.e. the creation, deletion, and modification, of a lifecycle of a cloud application or service. The orchestration engine abstracts various functions, and facilitates the management of resource lifecycles.

An application deployment consists of various pluggable modules, such as network functions (NF) that must be connected together. These pluggable functions are also referred to as virtual network functions (VNF). NFV supplies modules, that traditionally were provided to consumers as priority boxes, as processes in virtual machines (VM). In general, an application may want to specify a service graph (SG), that defines traffics order of traversal through a set of VNFs [3]. An SG includes a set of Service Level Agreements (SLA) that the infrastructure must fulfill. To satisfy these SLAs, the VNFs must be connected virtually, i.e. service chained. Chaining refers to the modification in configuration of network components such as switches, to direct packets through the set of intended modules.

This paper is organized as follows. Section 2 describes how we perform service chaining in an SDI environment. Section 3 describes what orchestration is and how we leverage an orchestration engine to facilitate service chaining. Section 4 considers our experiments with service chaining and their evaluation. Finally, the conclusion is presented in Sect. 5.

2 Service Chaining in SAVI

In this section we provide a more concrete definition of service chaining. We then consider the features provided by the SDI RMS, and how these can facilitate service chaining.

An application deployment consists of services that the end-user interacts with, e.g. a web server, and other NFs, i.e. transparent components like a load balancer. NFV refers to the virtualization of arbitrary NFs, such as deep packet inspection (DPI) firewalls, load balancers, etc. Individual NFs can be composed into a SG, that specifies a list of services and the order of traversal. For instance, consider a web application (app) deployment. This app may consist of a firewall that filters the traffic and a load balancer that distributes the load across horizontally scaled web servers. The SG is an abstract object that corresponds to

a set of SLAs. The realization of a SG is service chaining. Service chaining consists of two parts: creating the VNFs specified in the SG, and chaining them together. These can be done through the orchestration engine and the SDI manager, respectively. Specifically, the SDI manager has state information of the infrastructure and can direct the resource controllers to execute certain operations (See Fig. 1). These combined, allow the SDI manager to perform functions such as fault tolerance and dynamic installation of network flows.

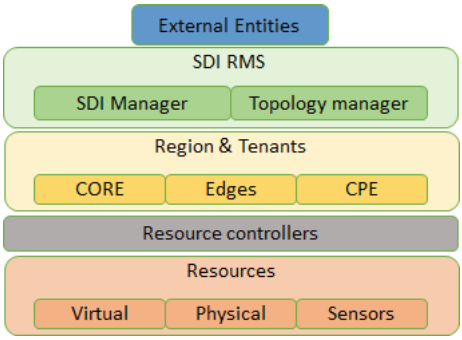

Fig. 1. SDI overall architecture

To facilitate service chaining, the SDI manager exposes many primitive functions, such as: tapping and blocking. Tapping refers to sending a copy of the incoming traffic to a host that was not the intended destination. Blocking concerns dropping packets according to system requirements in switches.

As an example of chaining, lets consider a WordPress deployment consisting of two VMs- one running a web server and the other running a database. It is desirable to allow the application to dynamically change their SG. For instance, the application may want to insert an inline deep packet inspection (DPI) VNF in front of the web server. The applications requirement can be satisfied using service chaining. Now lets consider how service chaining in the web server example could be realized. First, the application would request the SDI manager to perform service chaining by inserting a DPI VNF in front of the web server. The SDI manager would direct the network controller to install special high-priority flows in the switches of the underlying infrastructure to ensure all traffic headed for the web server goes to the DPI VNF. In this case we would have to configure the DPI unit to forward the traffic to the web server. Alternatively, we could use tapping so that the traffic is sent to the web server and mirrored to the DPI VNF [9]. Since resource controllers can be directed to execute commands at any time, service chaining can be performed on a live system without service disruption.

Multicasting is another example of an application that can leverage VNF chaining. Lets consider the sequences of events when deploying a multicasting

application. First, the orchestration engine creates casting modules such as virtualized transceivers and load balancers. Migrating the chained casting modules could reduce total bandwidth usage by reconstructing a more efficient multicast tree. In [10], Zhang et al. demonstrated cost reductions in deployments of multicast trees with a newly proposed routing algorithm that used dynamic chaining of VNFs.

3 Integration with Orchestration

Orchestration is the first step for service chaining. The challenge in service chaining consists of integrated management of multiple resource types such as computing and networking resources. First, we have to create the VNFs (these are typically VMs configured to perform the specific function). Second, we need to connect the nodes to allow the required communication. As described above, the SDI manager applies network policies (such as chaining) that ensure that the traffic traverses the required VNF(s). In our previous DPI example, first a DPI unit is created and then it is chained. Chaining has to position the firewall in such a way that all packets that are intended to reach the web server have to be redirected from the gateway through the firewall and to the web server.

Applications can request resources using an orchestration service on an SDI environment. For our experiments we utilized the Heat Orchestration project [4] from OpenStack. Heat facilitates the management of the creation, modification, and deletion of cloud infrastructure resources over the applications life cycle. Applications specify what resources they need and how they should be configured in descriptive template files. The orchestration engine then parses these templates to provision and configure the required resources. The applications can also modify and delete resources by providing new or modified templates to the orchestration engine. Therefore, the orchestration service allows management of complex topologies without increasing the cost of managing that complexity. Furthermore, template files for Heat are compatible with Amazon Web Services (AWS) CloudFormation (another orchestration service).

4 Implementation and Evaluations

We have conducted our service chaining experiments on the SAVI testbed to prove our concepts. The SAVI TB is an experimental platform intended to allow investigation into future application infrastructures. The SAVI TB is based on the SDI resource management architecture [1,5].

We conducted the following experiment to demonstrate dynamic service chaining. Our experiment consisted of a WordPress deployment and a DPI unit that were subsequently chained. We initiated two attacks to a web server and then attempted to block the attacks via a network intrusion detection and prevention system (NIDPS) that blocks the attackers in the same IDS. For both tests, the initial setups consisted of a web server (WS) running WordPress, and a database (DB) running MySQL. Both WS and DB were created using a Heat

orchestration template on the SAVI TB. The WS and DB were configured to allow the WordPress contents to be stored on the DB. This version of WordPress was a generic template representing a typical blogging web site over the Internet. Hence, it did not have any advanced filtering techniques embedded in itself.

We used a web user interface (Web UI) to allow users to request to chain the NIDPS between the client (potential attacker) and the web server. This web UI allows a user to conveniently login with credentials and select their SAVI node and project [5]. After login, the user could observe and select intended VMs for chaining operation and apply the modifications on the fly. The web UI sends the users request to the SDI manager to apply a network policy that steers the web traffic destined for the WordPress web server through the DPI.

Hence, to block the attack we utilized the NIDPS between attacker and the WordPress. Snort [8] is a highly active open source project that has been widely employed as a DPI. It incorporates groups of learned network policies to judge the validity of a packet or a group of packets passing through it. In addition, Snort was configured to act as a NIDPS. Merely, creating a DPI does not block the attack; we needed to chain it between the attacker and the WS. We tested our chaining and orchestration processes on the following two attack scenarios. The goal in both are to provide access for normal users and block attackers from overloading or corrupting the web server.

4.1 URL Injection Attack

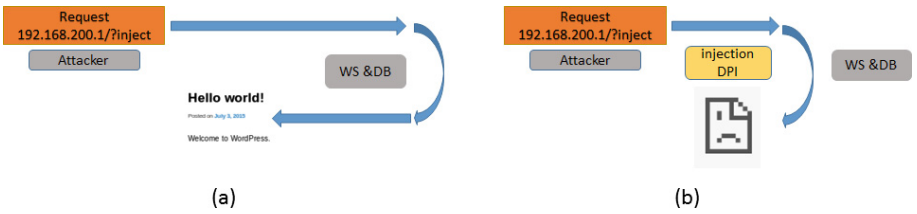

Fig. 2. URL injection attack

We first wanted to verify that attack would affect the web server without presence of a firewall. A normal user will insert correct address of the web server to the address bar, but an attacker would attempt to inject an extra text in the URL aiming to obtain or corrupt information on the Database (URL injection). Therefore, we initiated a sample URL injection attack on the web server (as seen in Fig. 2 part a), that passed an invalid argument to the WS. Specifically, we made malicious HTTP GET request by attaching a text at the end of the URL. Despite the malicious URL, the webpage still loaded, meaning the attack was successful. URL injection [7]could represent a general variety of access control or virus injection problems. To prevent such attacks we used the web UI to chain the firewall between the WS and the gateway. Afterward the same attack was

initiated. This time the DPI detected the attack and the web page didn't load. Hence the service chaining was successful (as seen in Fig. 2 part b).

4.2 DOS Attack

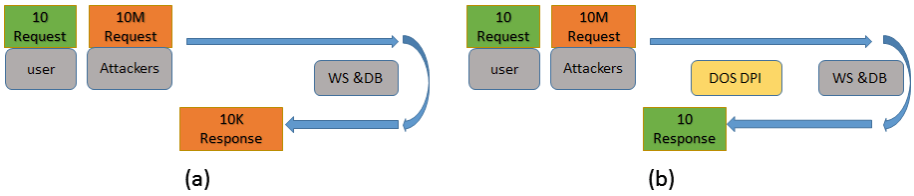

Fig. 3. DOS attack.

Our second test scenario was the detection and blockage of Denial of Service (DoS) attack. As upgraded networking and computing provide users with more bandwidth and processing, compromised users can more efficiently send high bandwidth (tens of Mbps) of requests to overload an attack target. This attack can be identified by deep inspection (i.e. using Snort) of packet headers that are being sent to a WS to identify malicious users. Specifically, malicious users send more than a threshold of requests that a web server could handle in a period of time. In [9] the network switch leading to the WS was tapped by the IDS. The IDS was processing the data and malicious attackers were reported to the SDI manager. The SDI manager subsequently blocked the malicious traffic at the ingress switch of the network. However, in this paper, where IDS is used as NIDPS, there is no longer a need to use SDI for blocking the attacker as Snort detects and blocks the attack right away.

We tested the WS without the firewall in place and initiated a DoS attack from multiple users. The attack made the web server slower and in some cases even inaccessible (as seen in Fig. 3 part a). Afterward by modifying the proper inputs to the web UI, a new DPI was placed in the middle of WS and the gateway. The new DPI was able to block the attackers from reaching the WS and protected the system against this DoS attack. Therefore the normal users were still able to access the WS even after the attack was initiated (as seen in Fig. 3 part b).

5 Conclusion

The SDI RMS allows converged management of networking and computing resources. These features, among other things allow us to easily perform service chaining. Service chaining consists of provisioning the required NFs and chaining them together. The SDI RMS exposes functions that allow us to apply

converged network and computing policies. Furthermore, we leverage the orchestration engine to provision and configure the VNFs. Using the orchestration service greatly reduces the complexity and effort in managing resource lifecycles. Service chaining and orchestration have eased development of complex architectures for hyper-dynamic applications. Therefore, employing instant placement modification of VNFs and dynamic application of network policies on SAVI testbed has brought SDI closer than ever to a promising autonomic platform.

References

1. Kang, J.-M., Lin, T., Bannazadeh, H., Leon-Garcia, A.: Software-defined infrastructure and the savi testbed. In: 9th International Conference on Testbeds and Research Infrastructures for the Development of Networks and Communities (TRIDENTCOM 2014), Guangzhou, People's Republic of China (2014)
2. Kang, J.-M., Bannazadeh, H., Leon-Garcia, A.: Savi testbed: control and management of converged virtual ict resources. In: 2013 IFIP/IEEE International Symposium on Integrated Network Management (IM 2013), pp. 664–667, May 2013
3. Keeney, J., van der Meer, S., Fallon, L.: Towards real-time management of virtualized telecommunication networks. In: 2014 10th International Conference on Network and Service Management (CNSM), pp. 388–393, November 2014
4. Kumar, R., Gupta, N., Charu, S., Jain, K., Jangir, S.K.: Open source solution for cloud computing platform using openstack. Int. J. Comput. Sci. Mob. Comput. **3**(5), 89–98 (2014)
5. Lin, T., Park, B., Bannazadeh, H., Leon-Garcia, A.: Savi testbed architecture and federation. In: 1st EAI International Conference on Future Access Enablers of Ubiquitous and Intelligent Infrastructures (Fabulous), September 2015
6. Lin, T., Kang, J.-M., Bannazadeh, H., Leon-Garcia, A.: Enabling sdn applications on software-defined infrastructure. In: Network Operations and Management Symposium (NOMS). IEEE, pp. 1–7, May 2014
7. Mookhey, K., Burghate, N.: Detection of sql injection and cross-site scripting attacks. Symantec SecurityFocus (2004)
8. Roesch, M., et al.: Snort: lightweight intrusion detection for networks. In: LISA 1999, pp. 229–238 (1999)
9. Yasrebi, P., Monfared, S., Bannazadeh, H., Leon-Garcia, A.: Security function virtualization in software defined infrastructure. In: 2015 IFIP/IEEE International Symposium on Integrated Network Management (IM), pp. 778–781, May 2015
10. Zhang, S., Zhang, Q., Bannazadeh, H., Leon-Garcia, A.: Routing algorithms for network function virtualization enabled multicast topology on sdn. IEEE Trans. Netw. Serv. Manage. **PP**(99), 1 (2015)