Automated Synthesis of NFV Topology: A Security Requirement-Oriented Design

A H M Jakaria*, M. Ashiqur Rahman*, and Carol Fung†
*Department of Computer Science, Tennessee Tech University, Cookeville, USA
†Department of Computer Science, Virginia Commonwealth University, Richmond, USA
Email: *ajakaria42@students.tntech.edu, *marahman@tntech.edu, †cfung@vcu.edu

***Abstract*—Cyber defense today heavily depends on expensive and proprietary hardware deployed at fixed locations. Network functions virtualization (NFV) reduces the limitations of these vendor specific hardware by allowing a flexible and dynamic implementation of virtual network functions in virtual machines running on commercial off-the-shelf servers. These network functions can work as a filter to distinguish between a legitimate packet and an attack packet, and can be deployed dynamically to balance the variable attack load. However, allocating resources to these virtual machines is an NP-hard problem. In this work, we propose a solution to this problem and determine the number and placement of the VMs. We design and implement NFVSynth, an automated framework that models the resource specifications, incoming packet processing requirements, and network bandwidth constraints. It uses satisfiability modulo theories (SMT) for modeling this synthesis problem and provides a satisfiable solution. We also present simulated experiments to demonstrate the scalability and usability of the solution.**

***Index Terms*—NFV architecture; formal modeling; DDoS security; synthesis.**

I. INTRODUCTION

Information security is one of the topmost priorities for any organization. Some recent incidents prove that different types of attacks are becoming stronger and more frequent day by day. Many of the solutions to these threats are composed of proprietary hardware. Upgrading or adding new network functions typically enforces the integration of more of these hardware appliances which requires time and imposes high costs. The traditional methods of threat detection are limited by the restricted computation capacity and inflexibility of involved network functions in dedicated hardware, such as firewall, router, intrusion detection systems (IDS), etc.

In NFV technology, network functions are implemented and deployed as virtual machines (VMs) in the form of software that runs on the commodity hardware. The VMs run on these general purpose hardware systems, which not only provides the benefit of elasticity, but also reduces the cost by running on low-cost commodity platforms like x86- or ARM-based servers instead of specialized hardware. NFV allows testing new apps more easily and offers an improved flexibility in assigning virtual network functions (VNFs) to hardware.

Utilizing VNFs to defend against cyberattacks has become a common trend these days. Rebahi *et al.* designed and developed a virtual security appliance (vSA) that is capable of detecting various network attacks while offering an acceptable level of performance [1]. A cloud-based architecture [2], and VGuard [3], a tool based on NFV, have been developed essentially to counter DDoS attacks. Liyanage *et al.* introduced NFV-based security apps to protect the LTE architecture [4]. Pastor *et al.* proposed use cases for an open operation, administration, and management (OAM) interface to virtualized security services for home/residential network access [5]. However, most of them fail to address the issue of resource management when deploying the associated VMs. Fayaz *et al.* [6] focused on an ISP-centric deployment model in Bohatei, where an ISP offers DDoS-defense-as-a-service to its customers by deploying multiple datacenters, and each datacenter has commodity hardware servers to run standard VNFs.VNGuard, a framework proposed by Deng *et al.* [7], performs management of virtual firewalls that protect virtual networks (VNs).

The physical properties of the servers, such as memory, CPU, etc., determine the capabilities of the VNFs running on the VMs within these servers [8]. Using the available server resources efficiently is a challenge because they are limited. The optimal placement of VMs in a network has been discussed in the literature [9], [10], [11]. Although the authors in these works formalized their approach considering several network resource constraints, as well as, service level agreements, they do not apply the idea to a security mechanism utilizing the VNFs [12], [13], [14]. A network architecture with the help of formal model was introduced in [6]. But it did not solve this problem in a timely and responsive manner. In this work, we present a novel tool, NFVSynth, which solves this bin packing problem using formal verification. It is an automated framework for synthesizing virtual network configurations and placements of VMs, using SMT constraint satisfaction checking.

II. NFVSYNTH FRAMEWORK

NFVSynth follows a top-down approach for the automation of NFV network architecture design synthesis. Fig. 1 presents an overview of the NFVSynth framework. The framework formally models the COTS server resource configurations, as well as the specifications of VMs. The packet processing requirements (*i.e.*, the attack traffic properties), the physical network topology including bandwidths of the links and latency of the servers from ingress points are also modeled by the framework. The framework formalizes the NFV architecture design synthesis problem as a VNF deployment plan that includes

978-3-901882-98-2 © 2017 IFIP

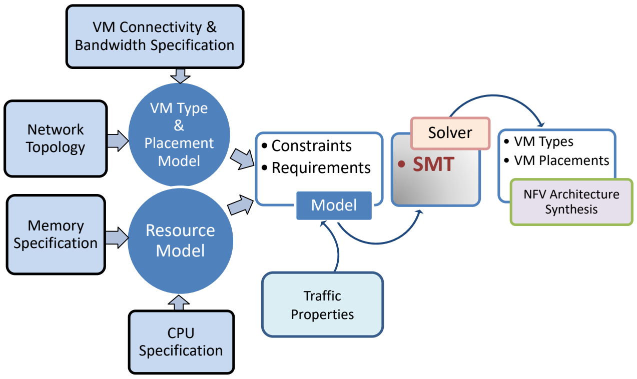


Fig. 1. The framework architecture of NFVSynth.

the determination of VM placements and properties (*e.g.*, type, memory, and CPU), satisfying the packet processing requirements within the physical resource constraints. Finally, it encodes the synthesis problem into SMT logics and provides a feasible solution using an efficient solver.

Use Case-based Design: Fayaz *et al.* [6] proposed a flexible and elastic DDoS defense system, Bohatei, that shows the benefits of software defined networking (SDN) [15] and NFV in the context of DDoS defense. It makes the use of NFV capabilities to elastically alter the required scale (*e.g.*, 10 Gbps vs. 100 Gbps attacks) and type (*e.g.*, SYN proxy vs. DNS reflector defense) of DDoS defense realized by defense functions running on VMs. A client-server model that uses packet filtering methods to disallow attack traffic from a client to reach the server was discussed in [16] by Jakaria *et al.* The defense filtering network is an NFV network that consists of dynamically created VNFs. In the physical layer, there are one or more product servers that provide online services to customers from the Internet, and some commodity servers that are connected to each other. Each server hosts VMs to realize different types of VNFs, such as dispatchers, switches, and agents. The dispatcher functions as the gateway for packets to the system. It runs a load balancing algorithm and distributes the incoming traffic to the deployed agents. The VNFs are organized in a way so that attack flows will be handled by filtering agents and they will filter out the attack traffic by performing a spoofed handshake with the client. The number of VNFs and their capability of processing packets depend on the commodity servers' available resources. The placement of the dispatcher and the agents, and their deployment decision are challenges that need to be solved.

We build our tool based on the preceding technique. In this case, the network topology of servers, their resources, and network bandwidth, as well as the attack traffic intensity, are provided to the model as input from a text file. The output from the SMT solver provides the count and placement of VMs implementing dispatchers and agents. NFVSynth can provide quick solutions to the problem based on the traffic changes.

III. NFVI Virtual Network Synthesis

This section discusses the formal model of the requirements and the constraints of the NFV architecture.

A. Packet Processing Requirement Model

In our design, we consider two types of VNFs: dispatcher and agent. All the incoming packets are forwarded to the dispatchers. Hence, a dispatcher requires high memory and CPU, which allow it to process high volume of attack traffic, as well as normal traffic. It should be installed on a server that can provide required resources to ensure that the dispatcher can efficiently dispatch the incoming packets to the agents and is not overwhelmed by the large number of packets. If $\mathbb{T}$ is the set of all the types, in our case, $\mathbb{T} = \{\mathrm{D}, \mathrm{A}\}$.

1) Resource Requirement Model: Memory and CPU cycles of the COTS servers are one of the main resources that we consider in this research.Let $M^V_{i,j,t}$ be the memory of VM j of type t running on server i; while $C^V_{i,j,t}$ be the CPU of that VM. The sum of the memories of all the deployed VMs on a server should be between a minimum and a maximum threshold percentage of the actual physical memory of that server which is available for the VMs. If μ is the maximum allowed percentage of memory utilization and $\bar{\mu}$ is the minimum of that, then the following should hold:

$$\forall_{i \in \mathbb{S}} \; (\sum_{j,t} M^V_{i,j,t} \leq \mu \times M^S_i) \wedge (\sum_{j,t} M^V_{i,j,t} \geq \bar{\mu} \times M^S_i) \quad (1)$$

The sum of the CPUs of all the deployed VMs on a server should not exceed the actual physical CPU of that server.

$$\forall_{i \in \mathbb{S}} \sum_{j,t} C^V_{i,j,t} \leq C^S_i \quad (2)$$

Regardless of the type of a VNF, the VM that contains it, must be allocated enough memory and CPU so that it has the best possible packet processing capabilities. The packet processing rate of a VM depends on the memory and CPU of the VM. If $P_{i,j,t}$ refers to the packet processing rate of a VM, we can express this as a function of memory and CPU, where α_t is a constant that determines the impact of memory and β_t is a constant determining the impact of CPU on the packet processing rate for a particular type, and $T^V_{i,j}$ determines the type of VM j on server i, which is of dispatcher and agent type in our case.

$$\forall_{t \in \mathbb{T}} \; (T^V_{i,j} = t) \; \rightarrow \; P_{i,j,t} = (\alpha_t \times M^V_{i,j,t}) \times (\beta_t \times C^V_{i,j,t}) \quad (3)$$

If a VNF is deployed on a VM, it needs to have memory and CPU greater than a minimum value. $\mathrm{M}_\mathrm{t}^\mathrm{min}$ and $\mathrm{C}_\mathrm{t}^\mathrm{min}$ refer to these minimum values for type t, respectively. This ensures that the packet processing rate depends both on memory and CPU. If $D^V_{i,j}$ denotes if VM j is deployed on sever i, the following holds:

$$\forall_{t \in \mathbb{T}} \; (T^V_{i,j} = t) \; \wedge D^V_{i,j} \; \rightarrow \; M^V_{i,j,t} \geq \mathrm{M}_\mathrm{t}^\mathrm{min} \quad (4)$$

$$\forall_{t \in \mathbb{T}} \; (T^V_{i,j} = t) \; \wedge D^V_{i,j} \; \rightarrow \; C^V_{i,j,t} \geq \mathrm{C}_\mathrm{t}^\mathrm{min} \quad (5)$$

2) Network Bandwidth Requirement Model: We take the overall network topology of the system as an input to our solver. The bandwidth of each link in the topology is also provided. It is required that the packet processing rate of the VMs does not exceed the bandwidth of the physical links, otherwise it would be impossible for the VMs to communicate with each other.

We denote the reachability between two VMs running on two VMs by $R_{i,j,k,l}$, where VM j is running on server i and VM l is running on server k. Each deployed agent should be reachable from the dispatcher, and vice versa. No communication is required between the agents only.

$$D^V_{i,j} \wedge D^V_{k,l} \wedge (T^V_{i,j} = \mathrm{D}) \wedge (T^V_{k,l} = \mathrm{A}) \\ \rightarrow\ R_{i,j,k,l} \tag{6}$$

$B^V_{i,j,k,l}$ is the required virtual bandwidth between two deployed VMs. Although there is a two-way communication between the dispatcher and an agent, we do not simply add up the packet processing rates of the communicating VMs to get the bandwidth. Instead, we double the processing rate of the agent. This is because, by design, the dispatcher will not forward more packets to an agent than it can handle. The dispatcher may need a higher bandwidth from the ingress point to itself, but it distributes the traffic to several agents which lessens the requirement of higher bandwidth on the other side. Thus, the required bandwidth between these two VMs should be, at least, twice as much as an agent can process. Consequently, if two VMs do not need to communicate, the virtual bandwidth between them should be zero.

$$R_{i,j,k,l}\ \rightarrow\ B^V_{i,j,k,l} \geq\ \min\{(2 \times P_{k,l}), (P_{i,j} + P_{k,l})\} \tag{7}$$

$$\neg R_{i,j,k,l}\ \rightarrow (B^V_{i,j,k,l} = 0) \tag{8}$$

The bandwidth of each link on the path from the server implementing the dispatcher and the server implementing the agents should be sufficiently high so that the VMs in them can talk to each other. $\mathbb{Z}$ is the set of all links on the path from a server to another server. $B^P_{i,k,z}$ is the physical bandwidth of z^{th} link on the path from server i to server k. The physical bandwidth of each link should be no less than the virtual bandwidth required by all the communicating VMs that are using that link. This will ensure the required throughput for the communication among the VMs. We model the constraint as the following, considering δ as the percentage of the physical bandwidth that a user would like to allocate for virtual communication:

$$R_{i,j,k,l}\ \rightarrow\ \forall_{z\in\mathbb{Z}}\forall_{i,k} \sum_{j,l} B^V_{i,j,k,l} \leq \delta \times B^P_{i,k,z} \tag{9}$$

B. Agent and Dispatcher Specific Requirement Model

The following constraint ensures that if a VM is deployed, it is either a dispatcher or an agent.

$$D^V_{i,j}\ \rightarrow\ (T^V_{i,j} = \mathrm{D}) \vee (T^V_{i,j} = \mathrm{A}) \tag{10}$$

$$\neg D^V_{i,j}\ \rightarrow\ (T^V_{i,j} \neq \mathrm{D}) \wedge (T^V_{i,j} \neq \mathrm{A}) \tag{11}$$

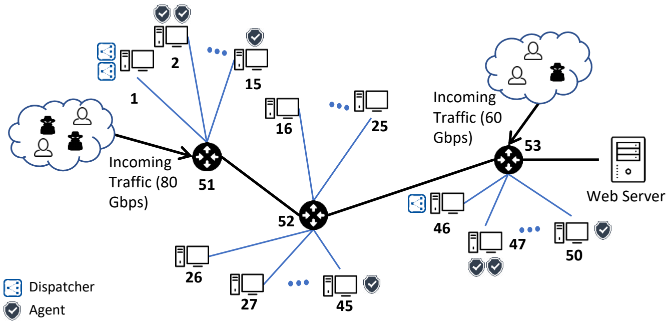


Fig. 2. The physical network topology of the servers and VMs implemented in them.

The combined packet processing rate of all the agents should be no less than the incoming packet rate. Let $P^{\mathrm{A}}_{i,j}$ be the processing rate of j^{th} agent located on server i. If P is the total number of ingress points and R_p is the incoming packet rate at ingress point p, then the following holds:

$$(T^V_{i,j} = \mathrm{A})\ \rightarrow\ \sum_{i,j} P^{\mathrm{A}}_{i,j} \geq \sum_{p=1}^{P} R_p \tag{12}$$

The total packet processing rate of all the dispatchers assigned for an ingress point should be at least equal to the incoming packet rate at that ingress point.

$$(T^V_{i,j} = \mathrm{D})\ \rightarrow\ \sum_{i,j} P^{\mathrm{D},p}_{i,j} \geq R_p \tag{13}$$

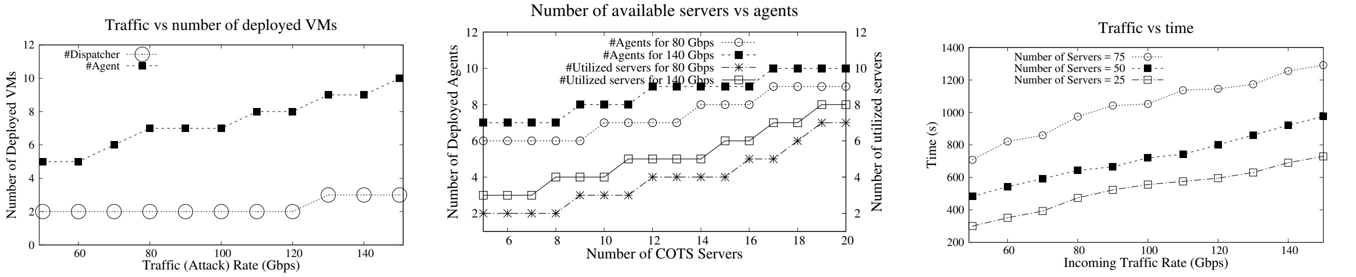
We consider the latency of each server from the ingress points, is a function of number of hops and the propagation delay, as a metric to choose servers for dispatchers. The server that is closest to an ingress point should be chosen to deploy the corresponding dispatcher. In case the ingress points are in closer proximity, we may consider deploying one dispatcher for multiple ingress points. In that case, we maintain a threshold latency (L_{th}) that is greater than the latency between an ingress point and a server.

$$\forall_{p\in\mathbb{P}}\ \exists_{i\in\mathbb{S}}\ (T^V_{i,j} = \mathrm{D})\ \rightarrow\ (L_{p,i} \leq\ L_{th}) \tag{14}$$

The synthesis problem is formalized as the satisfaction of the conjunction of all the constraints in Equations 1 through 14. We implement our model by encoding the system configuration and the constraints into SMT logics. In this encoding purpose, we use the Z3, an efficient SMT solver [17]. The solver checks the verification constraints and provides a satisfiable (SAT) result if all the constraints are satisfied.

IV. A Synthetic Case Study

A synthetic network of commodity servers is shown in Fig. 2. In this example, there are 50 COTS servers in the NFV network that are connected to each other. From an input text file, we read the physical connectivity of these servers through routers in terms of the bandwidth of links and latency from the ingress points. Memory is provided in GB and CPU is in number of cores. We consider two ingress points, each receiving 80 Gbps and 60 Gbps of traffic respectively.



(a) (b) (c)

Fig. 3. (a) Number of required VMs w.r.t. incoming traffic rate, (b) number of deployed agents and utilized servers w.r.t. total available COTS servers, and (c) model synthesis time w.r.t. incoming traffic rate

NFVSynth gives a SAT result for this example. From the resultant SAT instance, we find the deployed VM types ($T^V_{i,j}$) along with their placements ($D^V_{i,j}$) in the servers. The result shows that three dispatchers need to be deployed on server 1 and 46, while several agents need to be installed on server 2, 15, 45 and so on. The dispatchers are deployed on servers that are closer to the ingress point in terms of latency. There are only three dispatchers required, because they only need to dispatch the whole incoming traffic, be it legitimate or an attack, to all the deployed agents. The packet processing rate of the dispatcher is, at least, equal to the incoming packet rate at its corresponding ingress point, while the combined packet processing rate of the agents is more than the total incoming rate. In our case, communication is required between a dispatcher and an agent; virtual bandwidth between any two dispatcher and agent is in accordance with the bandwidth of the physical links. NFVSynth also provides the required memory, CPU and the packet processing rate of each VM.

V. Evaluation

In our evaluation, we first present the analysis on the relationships among different deployment parameters such as traffic and resources. Then, we present the performance (*i.e.*, scalability) analysis of the tool. To evaluate NFVSynth, we ran experiments on different network topologies of different configurations and connectivity of $5-100$ COTS servers. The servers are equipped with memory between $16-48$ GB and CPU cores of $2-7$. NFVSynth was run on a machine running Windows 10 OS. The machine is equipped with an Intel Core i7 Processor and a 16 GB memory.

A. Relationships between Deployment Parameters

In this analysis, we ran a number of experiments on similar network topologies.We increased the traffic rate, which includes the attack packets, gradually from 50 to 150 Gbps, and observed the number of deployed dispatchers and agents. This is demonstrated in Fig. 3(a). In this case, with the increment of traffic, the number of dispatchers increases very slowly, in comparison to the number of agents. It is possible for the same number of agents to process a certain range of traffic rate. In the figure, the number of agents remains the same between traffic rates of 80 and 100 Gbps. As the traffic rate passes beyond 100 Gbps, the number of agents increase to 8 from 7; it remains the same up to 120 Gbps.

Fig. 3(b) shows the number of deployed VMs, as well as the number of utilized servers, with respect to the number of total available servers in the network topology for a certain amount of incoming traffic (80 and 140 Gbps). As the number of available servers increases, the number of agents also increases slowly. The resource and bandwidth constraints are responsible for this, as NFVSynth tries to find a solution utilizing all the prospective VMs. The number of candidate servers for deploying VMs increases in the system as there are more servers. The graph demonstrates that the number of required agents and utilized servers for 140 Gbps of traffic is higher than that of 80 Gbps of traffic. We can also observe from the graph that the increasing rate of number of utilized servers is higher than the rate of increasing agents.

B. Scalability

We evaluate the scalability of NFVSynth by analyzing the time required to synthesize the virtual topology by varying the problem size. The synthesis time includes the model generation time and the constraint verification time. The model synthesis time with respect to the incoming traffic rate is shown in Fig. 3(c). Three scenarios with 25, 50 and 75 servers are presented in the graph. We observe that the required time increases almost linearly with incoming traffic rate. As the attack traffic increases, added VMs need more resources to process the traffic. As a result, the constraints become more strict to solve. As a result, the solver takes more time to find a satisfiable solution.

VI. Conclusion

NFVSynth is tool for the recent networking trend, NFV, which is used widely in different cyber defense techniques. We deal with challenges in allocating resources to VMs that implement virtual network functions. The tool formally models the defense requirements and resource constraints, and formalizes the NFV architecture synthesis problem. Then, it solves the problem using an efficient SMT solver that results in the placements and classifications of the VMs within a sustainable period of time.

References

[1] Y. Rebahi et al. Virtual security appliances: the next generation security. In *2015 Int. Conf. on Communications, Management and Telecommunications (ComManTel)*, pages 103–110. IEEE, 2015.

[2] S. Yu et al. Can we beat DDoS attacks in clouds? *IEEE Transactions on Parallel and Distributed Systems*, 25(9):2245–2254, 2014.

[3] C. J. Fung and B. McCormick. VGuard: A Distributed Denial of Service attack mitigation method using Network Function Virtualization. In *CNSM, 11th International Conference on*, pages 64–70. IEEE, 2015.

[4] M. Liyanage et al. Leveraging LTE security with SDN and NFV. In *2015 IEEE 10th Int. Conf. on Industrial and Information Systems (ICIIS)*, pages 220–225. IEEE, 2015.

[5] A. Pastor and D. Lopez. Access Use Cases for an Open OAM Interface to Virtualized Security Services. 2014.

[6] S. K. Fayaz, Y. Tobioka, V. Sekar, and M. Bailey. Bohatei: Flexible and elastic ddos defense. In *24th USENIX Security Symposium*, pages 817–832. USENIX, 2015.

[7] J. Deng et al. VNGuard: An NFV/SDN combination framework for provisioning and managing virtual firewalls. In *NFV and SDN, IEEE Conference on*, pages 107–114. IEEE, 2015.

[8] N. Egi et al. Understanding the packet processing capability of multi-core servers. Technical report, Intel Technical Report, 2009.

[9] B. Addis et al. Virtual network functions placement and routing optimization. In *Cloud Networking (CloudNet), 2015 IEEE 4th International Conference on*, pages 171–177. IEEE, 2015.

[10] S. Mehraghdam et al. Specifying and placing chains of virtual network functions. In *Cloud Networking (CloudNet), 2014 IEEE 3rd International Conference on*, pages 7–13. IEEE, 2014.

[11] M. Bari et al. On orchestrating virtual network functions. In *Network and Service Management (CNSM), 2015 11th International Conference on*, pages 50–56. IEEE, 2015.

[12] M. Chowdhury et al. Implementation and performance analysis of various vm placement strategies in cloudsim. *Journal of Cloud Computing*, 4(1):20, 2015.

[13] M. Masdari et al. An overview of virtual machine placement schemes in cloud computing. *Journal of Network and Computer Applications*, 66:106–127, 2016.

[14] R. Cohen et al. Near optimal placement of virtual network functions. In *Computer Communications (INFOCOM), 2015 IEEE Conference on*, pages 1346–1354. IEEE, 2015.

[15] B. Nunes et al. A survey of software-defined networking: Past, present, and future of programmable networks. *Communications Surveys & Tutorials, IEEE*, 16(3):1617–1634, 2014.

[16] A. Jakaria et al. VFence: A Defense against Distributed Denial of Service attacks using Network Function Virtualization. In *STPSA, 11th IEEE International Workshop*. IEEE, 2016.

[17] L. D. Moura and N. Bjørner. Z3: An efficient SMT solver. In *Conf. on Tools and Algo. for the Construction and Analysis of Systems*, 2008.