The Virtual Network Function Placement Problem

Xin Li and Chen Qian
Department of Computer Science, University of Kentucky
Email: xin.li@uky.edu, qian@cs.uky.edu

I. Introduction

Network functions are widely deployed in modern networks, providing various network services ranging from intrusion detection to HTTP caching, for the purpose of performance, security or policy compliance. According to a recent survey [6], network function deployments are ubiquitous: on par with the number of L3 infrastructures. Application or user requirements may specify policies that require flows traverse through a give sequence of networks functions, called policy enforcement.

For example, the network administrator may specify a policy that all http traffic should follow the **policy chain**: *firewall* $\rightarrow$ *IDS* $\rightarrow$ *proxy*.

Traditionally, the network operators adopt commercial hardware middleboxes to provide network functions. The hardware middlebox introduces large capital expenses as well as expensive operational costs. As a result, only limited number of hardware middleboxes are deployed in a network. Once middleboxes are deployed, it is unfeasible to redeploy these hardware devices. To enforce polices, *traffic steering* is utilized, to make changes on flow paths such that flows can traverse through the specified middleboxes. However, traffic steering suffers from the following problems:

- Routing path changes may affect the performance of other control applications such as traffic engineering. Ideally, policy enforcement should be completely orthogonal to other applications.
- Traffic steering introduces additional path length , which increases the probability of congestion [8]. The links near middleboxes are likely to become hot spots. The situation is aggravated if the middleboxes are not properly placed in the network.
- Forwarding loops may be caused by traffic steering.

Recently, both academia and industry are studying Network Function Virtualization (NFV) [3], a technique that proposes to replace hardware middleboxes with software-based network functions running on generic platforms, such as X86. With NFV, a new but intuitive policy-enforcement scheme is proposed in this abstract, which overcomes the aforementioned drawbacks.

This new policy-enforcement scheme, proposes to actively place the required Virtual Network Function (VNF) instances on the path of each flow, rather than alter flow paths. One example is depicted in Fig. 1. An SDN-compatible central controller can install VNF instances on the hosts connected to the routers on the path. The overhead associated with the one-

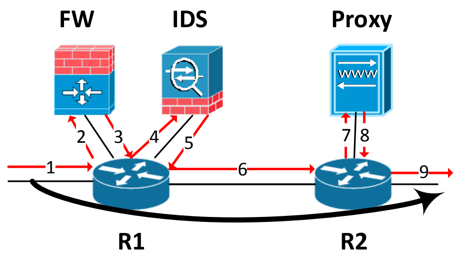

Fig. 1. The policy for the flow is *firewall* $\rightarrow$ *IDS* $\rightarrow$ *proxy*. $R1$ and $R2$ are on the path of the flow.

hop indirect path is insignificant compared to traffic steering, especially in low-latency data center networks. As a result, our policy-enforcement scheme does not influence control plane, and thus being completely orthogonal to applications such as traffic engineering. The virtual network function instances can be easily allocated/removed just like virtual machine placement and migration. Some network functions can also be processed on the router or switch [9].

To enforce policies in our scheme, the requirements are two-folded for each flow:

1) For each network function specified by the policy for a flow, at least one instance is on the network path.
2) If 1) is satisfied, for any VNF instance n, there should be at lease one instance of the VNFs succeeding n on the same router of n or the understream routers on the flow path.

II. Problem Statement

In this abstract, we would like to answer the following question in the context of our policy-enforcement scheme:

How can network function instances be placed, such that polices are enforced with minimal resources consumption?

For NFV, the network functions consume hardware resources (such as CPU, memory, disk, I/O) in the generic platform (such as server) to launch a new instance. In this abstract, we only consider hardware consumption. As a first step to solve this problem, we are seeking a way to enforce polices with minimal number of network function instances. More complex considerations (network cost, heterogeneity in resources consumption) will be in our future work.

The assumption in the question is that network function can be shared among different flows. However, it depends from case to case, especially for the cloud environment, where the tenant networks need to share underlying infrastructure, but use some exclusive network functions such as firewalls. From our observations, the network functions can be divided into three categories:

1) Network functions can be used only by one single tenant network. The traffic of each tenant network are intended to be isolated. Sharing the network function in the network may violate isolation. One example is web proxy. Suppose two tenant networks share the same proxy server. Each tenant network can get the cached version of the web pages. Security problem occurs if adversaries hiding in the other tenant network injecting malicious content into the cached web pages.
2) Network functions of the same configuration can be shared by different tenant networks, such as firewalls of two tenant networks belonging to the same organization.
3) Network functions can be shared by the whole cloud. Some network functions are stateless and do not need customized configuration. Deep Packet Inspection (DPI) based on packet signature [4] is one of many such network functions.

As a result, in our policy-enforcement scheme, some network functions are further broken ties by either tenant network or configuration.

A. Mathematical Formulation

The VNF placement problem can be formulated as a integer linear program (ILP).

The network topology is represented by a graph $G = (V, E)$, where V is the set of routers in the network. Denote as N a set of VNFs. Let $x_n^v \in \{0, 1\}$ indicate whether router $v \in V$ is connected to one VNF instance $n \in N$. There is a set of network flows F in the network. For each $f \in F$, the path is denoted as a sequence $P_f =< p_f^i >$, where $p_f^i \in V$. The associated policy chain is denoted as a sequence $C_f =< c_f^i >$, where $c_f^i \in N$. We introduce a new variable $\sigma_{f,j}^i$, which counts the number of instance of network function $c_f^j \in N$ on the flow path of $f \in F$, from beginning to p_f^i on the network path. Likewise, let $y_{f,j}^i \in \{0, 1\}$ indicate whether flow $f \in F$ is processed by VNF instance $c_f^j \in N$ connected to the p_f^i. The linear programming is:

$$Minimize \sum_{v \in V} \sum_{n \in N} x_n^v \qquad (1)$$

s.t.

$$\sum_i y_{f,j}^i \geq 1 \qquad \forall f, i, j \qquad (2)$$

$$\sigma_{f,j}^i = \sigma_{f,j}^{i-1} + y_{f,j}^i \qquad \forall f, i, j \qquad (3)$$

$$\sigma_{f,j-1}^i - \sigma_{f,j}^i \geq 0 \qquad \forall f, i, j \qquad (4)$$

$$x_{c_f^j}^{p_f^i} - y_{f,j}^i \geq 1 \qquad \forall f, i, j \qquad (5)$$

$$y_{f,j}^i \in \{0, 1\} \qquad \forall f, i, j \qquad (6)$$

$$x_n^v \in \{0, 1\} \qquad \forall n, v \qquad (7)$$

Method	Performance	Flow #			
		725	1500	2500	5000
CPLEX	Best Value	101	103	159	251
	CPU Sec	6	32	99	147856
SA	Best Value	108	111	165	270
	CPU Sec	1.19	2.31	5.47	14.74

TABLE I
DIFFERENT PERFORMANCE AGAINST VARIOUS NUMBER OF FLOWS

Eq. (2) means that at least one instance exists for each network function in the policy chain for each flow, which is the first requirement of our policy-enforcement scheme stated in Sec. I. Eq. (4) makes sure that the order of policy chain is enforced, as the second requirement in Sec. I.

B. Problem Complexity

Set Cover Problem (SCP) can be reduced to the VNF placement problem. Due the page limt, we skip the proof. Since SCP is a NP-hard problem, the formalized VNF placement problem is also NP-hard.

III. INITIAL METHOD AND RESULTS

We synthesize network function policies based on real-network study by [5]. The topology we use is RocketFuel 3967, the router-level ISP network topology of AS 3967 [7]. The traffic trace is from CAIDA [1].

We run IBM ILOG CPLEX Optimizer [2] to solve the linear programming problem mentioned in Sec. II-A to get the optimal solution on a quadcore@3.40G desktop with 16GB memory. As we seen in TABLE. I, this method using CPLEX to get the optimal solution is not scalable. On the other hand, we develop a Simulated Annealing (SA) based heuristic to solve optimization problems. Preliminary results in TABLE. I indicate that SA can get the approximate solution in much shorter time. Due to the page limit. We skip the detailed algorithm description.

REFERENCES

[1] The caida ucsd anonymized internet traces 2013 - 2014. mar. http://www.caida.org/data/passive/passive_2013_dataset.xml.
[2] Ibm ilog cplex optimizer. http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/.
[3] R. Guerzoni et al. Network functions virtualisation: an introduction, benefits, enablers, challenges and call for action, introductory white paper. In *SDN and OpenFlow World Congress*, 2012.
[4] M. Roesch et al. Snort: Lightweight intrusion detection for networks. In *LISA*, volume 99, 1999.
[5] V. Sekar, S. Ratnasamy, M. K. Reiter, N. Egi, and G. Shi. The middlebox manifesto: enabling innovation in middlebox deployment. In *Proc. of ACM HotNets*, 2011.
[6] J. Sherry and S. Ratnasamy. A Survey of Enterprise Middlebox Deployments. Technical report, EECS, UC Berkeley, 2012.
[7] N. Spring, R. Mahajan, and D. Wetherall. Measuring ISP topologies with Rocketfuel. 2002.
[8] Y. Yu and C. Qian. Space shuffle: A scalable, flexible, and high-bandwidth data center network. In *Proc. of IEEE ICNP*, 2014.
[9] Y. Yu, C. Qian, and X. Li. Distributed collaborative monitoring in software defined networks. In *Proc. of ACM HotSDN*, 2014.