LightChain: A Lightweight Optimisation of VNF Placement for Service Chaining in NFV

Anish Hirwe
Department of Computer Science and Engineering
Indian Institute of Technology Hyderabad
Email: cs14resch01001@iith.ac.in

Kotaro Kataoka
Department of Computer Science and Engineering
Indian Institute of Technology Hyderabad
Email: kotaro@iith.ac.in

***Abstract*—In network function virtualization (NFV), the placement of the virtual network function (VNF) significantly affects the load on switches and the efficiency of bandwidth utilization. Suboptimal placement of VNFs in service chains increases the flow rules in the switches and the ping-pong traffic among the VNFs. This paper presents *LightChain*, an efficient polynomial run time approach for optimizing VNF placement and service chaining in NFV. *LightChain* forms the directed acyclic graph of the given service chains and applies topological sorting to the graph. The significance of the proposed approach is as follows: 1) The minimum hop-count in each service chain, 2) Reduction of flow rules on SDN switches, and 3) Elimination of ping-pong traffic. Evaluation results show that *LightChain* can determine the placement locations of the VNFs in a reasonably short time. Our comparison results with other approach show that *LightChain* installs a lower number of flow rules in the switches.**

I. Introduction

In Network Function Virtualization (NFV), the services and network functions are implemented as Virtualized Network Functions (VNFs) [1]. The utilization and placement of VNFs can be managed according to their demand in the network, which changes dynamically with time. To apply the policy or provide network services to traffic, the traffic must go through a particular sequence of VNFs called a *service chain*. *Service chaining* [2] [3] [4] [5] refers to the approaches for ordering the sequence of VNFs and the enforcement, wherein the traffic must go through the proper service chain. In conventional networks, service chaining has been implemented through manual configuration [6]. However, in the case of NFV, VNFs can be dynamically created. Optimizing the placement of dynamically created VNFs and determining the best path for traversing VNFs in service chain are challenging problems. SDN allows us to view the network as a global entity, and a centralized controller provides programmable forwarding rules that reduce the complexity of service chaining enforcement [7] [8] [9] [10]. VNF Placement (a) in figure 1 shows an example of VNF deployment of service chains in a network. S1 and S2 forward the corresponding flow through the service chain, which includes *Proxy, Firewall*, and *IPTV Server*. In contrast, VNF Placement (b) shows another way, which has more forwarding rules and a longer path that must be traversed by the flow. A slight change in the sequence of VNF placement can introduce huge inefficiency in traffic handling between VNFs-say, ping-pong traffic- that results in wastage of network resources such as switch memory (TCAM) and network bandwidth. For a large network with a long and large number of service chains, this problem becomes significant. Even a small scale data center has millions of flows. Suboptimal placement of VNFs will reduce both the efficiency of resource utilization and the performance of a data-center.

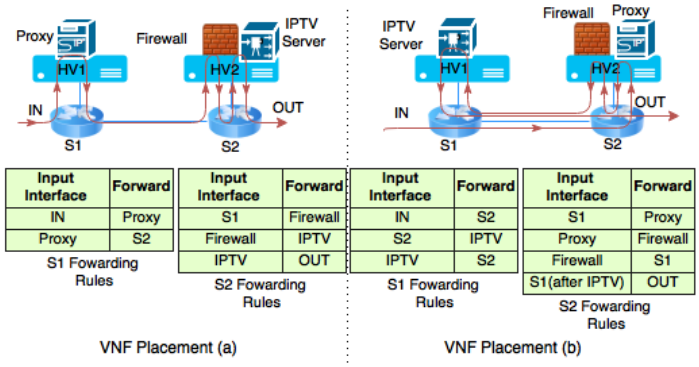

Fig. 1. Two different patterns of VNF placement for the same Service Chain $P = Proxy- > Firewall- > IPTV Server$

A. Cost of a Service Chain

Let P and M be the set of service chains and set of VNFs, respectively; $P = \{P_1, P_2, \ldots, P_n\}$ and $M = \{m_1, m_2, \ldots, m_k\}$. A service chain P_i is a sequence of VNFs M_i. Let V be the set of switches in a network topology T. The *Cost* C_i of a placement of M_i, for a service chain P_i, on T is defined as

$$C_i = D(V_{in}, m_1) + D(m_1, m_2) + \ldots + D(m_l, V_{out})$$

where V_{in} and V_{out} are the ingress and egress switches, and $D(m_r, m_t)$ is the hop count between m_r and m_t. The optimal placement of VNFs for a service chain P_i on the switches V has the minimal C_i. In this paper, we present a heuristic of VNF placement to reduce the number of flow rules on the switches and eliminate the ping-pong traffic. Our goal is to place the VNFs in the service chains efficiently to minimize the C_i of the service chains.

II. Related Work

There has been a significant amount of work in virtual machine placement problem in clouds and data-center networks [11] [12] [13]. Formalization of the network function placement and chaining problem was presented by Luizelli *et al* [14]. They proposed an integer linear programming-based optimization model to solve the placement and chaining problem by decomposing the problem into three phases: Placement, Assignment, and Chaining. In these phases, they determine the locations of the VNFs, assign instances of VNFs to flow, and create paths that connect the VNFs. Theoretical analysis of the NFV location problem is modeled into two well-known NP-hard optimization problems: (1) the facility location problem and (2) the generalized assignment problem by Cohen *et al* [15]. They proposed approximation algorithms to solve the NFV location problem. Their goals are similar to ours. Moreover, our cost metric also considers the distance and hop-count. However, these papers do not consider the order of VNF placements. Thus, we propose a simple polynomial run time algorithm for VNF placement. Some research group have proposed the outsourcing of the middlebox processing in the cloud [16]. However, they do not discuss the placement of middleboxes in the cloud. Traffic control and coordination and middlebox load balancing are presented in [8] [6] [17] [10]. LightChain's traffic steering technique is inspired by their approaches.

III. Service Chaining in NFV Orchestration

Figure 2 gives an overview of the system and shows the interaction among the different components of the system. The system in the end places VNFs on the hypervisors and installs the flow rules on the SDN switch to steer the flows to VNFs.

A. SLA Manager and Control Application

The SLA Manager receives the policy from the Admin. Policies are represented by data-flow [6] [18] [19]. The Admin specifies the policies, which contain the service chaining information. For example, a service chain for student traffic in an educational institute is $Firewall => IDS$. The SLA Manager translates these logical policies into machine-readable form to be further processed by the Control Application. The Control Application is the brain of the network. It monitors the network and collects the state information of the network, *i.e.*, switch connection information, link utilization, etc. The Control Application manages the Orchestrator and Flow Manager.

B. Orchestrator

The responsibility of the Orchestrator is to find the locations that can place VNFs. It interacts with Topology Manager, which maintains the topological information about the network. The Topology Manager monitors and collects data from the network and hypervisors, *i.e.*, link utilization of the network, TCAM of the switches, location of the hypervisor, and load on the hypervisors. The Orchestrator runs our heuristic that determines the order of VNFs and feasible locations of the VNFs in the service chains. It also deploys VNFs on the hypervisors.

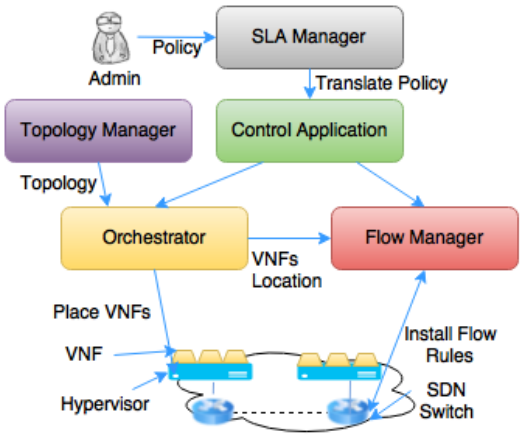

Fig. 2. System Design

C. Flow Manager

The Flow Manager steers the traffic from the source to the destination. It interacts with the Orchestrator to obtain the locations for VNF placement. The flow rules are then installed on the switches such that the flow traverse all VNFs of its service chain.

IV. Optimizing VNF placement in Service Chain

The cost C_i of VNF placement is optimal if the hop-count of all adjacent VNFs in a service chain P_i is minimal. This section describes the heuristics to optimize the VNF placement for all service chains using topological sorting to order the VNF sequence.

A. Ordering the VNFs

Let G be the directed graph of the set of service chains P with $|M|$ VNFs. There exists an edge from VNFs m_i to m_j if a service chain contains a VNF sequence from m_i to m_j. The goal is to find the topological ordering of the graph G. The topological order is the linear ordering of the vertices of a Directed Acyclic Graph (DAG) such that for all edges (u, v), u appears before v in the ordering [20]. First, the graph G should be acyclic because topological ordering is possible only for DAGs. A graph can be made acyclic by removing a random edge from the cycle. However, the topological ordering of a graph will not give the correct ordering of VNFs because the dependencies between the sequences of VNFs have been lost. Our approach of ordering the VNFs involves a two-step procedure; the first step creates the DAG of the service chains; the second applies the topological sort to the vertices of the DAG. Algorithm 1 shows the pseudo-code for the ordering of the VNFs.

1) Creating the DAG of the Service Chains: Figure 3 illustrates an example of creating DAG graph from the two service chains. Let P_1 and P_2 be the two service chains. In the first step, a simple directed graph of service chain P_1 and

Algorithm 1 Ordering the VNFs

Input: Set of Service Chains P

1: $G = createDirectedGraph(P)$
2: **if** G contain cycles **then**
3: $\quad G = removeCycle(G)$
4: **end if**
5: $C = FindConnectedComponents(G)$
6: **for** $i = 0$ to $length(C)$ **do**
7: $\quad T_{sort}(C_i) = TopologicalSort(C_i)$
8: **end for**
9: **return** $T_{sort}(C)$

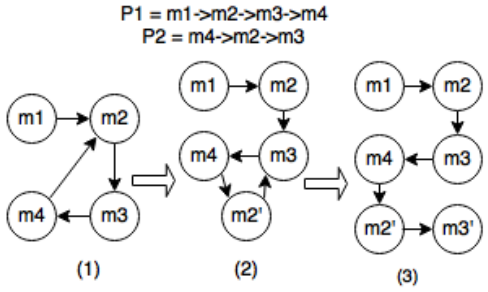

Fig. 3. Removing cycles from the directed graph

P_2 is created. Here, a chain $m4 => m2$ is creating a cycle. In the next step, another instance of $m2$ is created. Now, in chain $P2$, $m2$ becomes $m2'$. The graph is still not acyclic because a chain $m2' => m3$ is creating the cycle. Applying the same procedure again gives the new instance $m3'$. Now, the modified service chain $P2$ is $P2 = m4 => m2' => m3'$. Block (3) in figure 3 shows the final DAG of the service chains P_1 and P_2. It is clearly visible that the created DAG preserves all dependencies of the service chains.

2) Conducting Topological Sort on the DAG: Suppose G has connected components $C_1, C_2, \ldots, C_x$. The topological sort on the DAG G can be performed by repeatedly (1) finding a vertex of in-degree 0, (2) returning the vertex, and (3) removing the vertex and all its outgoing edges from the graph [20]. The topological sort on components $C_1, C_2, \ldots, C_x$ gives the order in which VNFs should be placed. The reason to apply the topological sort on individual components instead of the whole graph is that VNFs of one component are independent of other components. Block (3) in figure 3 has one connected component $C_1 = \{m_1, m_2, m_2', m_3, m_3', m_4\}$. The topological order of C_1 is $T_{Sort}(C_1) = \{m_1, m_2, m_3, m_4, m_2', m_3'\}$. There are different ways to place $T_{sort}(C_x)$ on a network topology. One is to find the path of length k (k is the total number of VNFs) or more than k on T, and place all VNFs on that path. However, finding the path of length k or more than k is the reduction of longest path problem which is an NP-compete problem [20]. The next section describes the shortest path heuristics to place VNFs on the network.

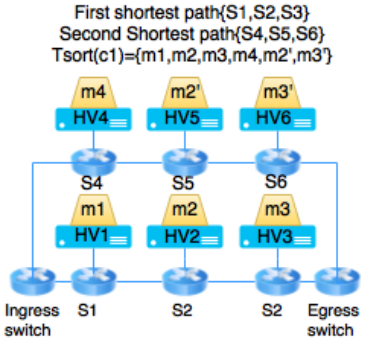

Fig. 4. Placement of the VNFs on the network

B. Placing the VNFs (Shortest path heuristics)

In the shortest path heuristics, VNFs of each component are placed on the shortest path between the ingress and egress switches of the network. The path length of n means that this path can host n VNFs. To minimize the number of forwarding rules in the switches, our heuristics place all VNFs on the shortest path. First, heuristics determine the shortest path between the ingress and egress switches using Dijkstra's algorithm. Here, the assumption is that the network is large enough to place all VNFs of service chains on the shortest path. If that path is exhausted, then the new shortest path is determined. In this case, all vertices included in the previous shortest path are removed from the graph. Algorithm 2 shows the pseudo-code for the placement of the VNFs on the shortest path. Figure 4 illustrates the placement of $T_{Sort}(C_1)$ on the network. Our assumption in this example is that each of hypervisors can accommodate only one VNF. The first shortest path is $S1, S2, S3$ with path length three being able to accommodate three VNFs. Here, $m1$ is placed on $H1$ because it is the first VNF of $T_{Sort}(C_1)$. We iterate through all VNFs of $T_{Sort}(C_1)$ and place it on the hypervisor one by one. After placing three VNFs, the first shortest path is exhausted. Therefore, the second shortest path must be found, which is $S4, S5, S6$. The remaining unplaced VNFs, $m4, m3', m2'$, can be placed on the second shortest path.

Algorithm 2 Shortest Path Placement

Input: $T_{sort}(C)$, Network Topology T, $ingress$ and $egress$ switch

1: $shortestPath = dijkstra's(T, ingress, egress)$
2: **for** $T_{sort}(C)$ has more VNFs **do**
3: $\quad$ **if** $shortestPath$ is not exhausted **then**
4: $\quad\quad Locations = (T_{sort}(C), shortestPath)$
5: $\quad$ **end if**
6: $\quad$ **if** $shortestPath$ is exhausted **then**
7: $\quad\quad removeShortestPathEdge(shortestPath, T)$
8: $\quad$ **end if**
9: **end for**
10: **return** $Locations$ {locations of the VNFs}

TABLE I
EXPERIMENTAL SETUP

#Switches	#chains	#VNFs	VNFs/Chain
100	40	200	5

C. Traffic Steering

The proposed heuristic places the VNFs in the topological order, and the VNFs of service chains do not form the loops. To install flow rules on the switches, the Flow Manager must know the locations of the VNFs on the network and the shortest paths on which VNFs have been placed. The Flow Manager interacts with the Orchestrator to obtain the shortest paths and the placement locations of the VNFs. The location is defined as the 4-tuple $(VNF\_id, Hypervisor\_id, Switch\_id, Switch\_port)$, where VNF_id: unique ID of the VNF, $Hypervisor\_id$: unique ID of a hypervisor that is hosting the VNF, $Switch\_id$: DPID of a switch that is attached to a hypervisor, and $Switch\_port$: physical port number of a switch that is connected to a hypervisor. The Flow Manager maintains a hash table to store locations of the VNFs where *VNF_id* is the key of the hash table. To generate the flow rule for a particular VNF_id, *Flow Manager* finds the location of VNF_id from the hash table. The flow rules are then installed in the switch based on Switch_id and Switch_port that are attached to a hypervisor and is hosting VNF_id.

V. EVALUATION

We simulate our heuristic to answer the following questions: 1) Can our heuristic place all VNFs in polynomial run time in a large network with a larger number of service chains? 2) What is the cost of service chains? 3) How does the VNF placement reduce the number of flow rules and ping-pong traffic? We evaluate the system on the data-center hierarchical topology with 100 edge switches. Table 1 shows the experimental setup of the system.

A. Run time of Orchestration

To verify the tractability of our algorithm, we examine the run time of the Orchestrator by increasing the hypervisors per switch and VNFs per hypervisor. Figure 5 shows the average computation time to determine the placement locations of the VNFs with respect to hypervisors per switch. The run time increases linearly instead of exhibiting an exponential curve. The result confirms the polynomial run time of our algorithm.

B. Cost of the Service Chain

We defined the *Cost* C_i of a service chain in Section 1 as the hop-count of a service chain. The *Cost* of the service chains depends on the VNF sequence in the service chains and VNF dependencies among the service chains. We compare *LightChain* with the *First Come First Place (FCFP)* approach. In FCFP, VNFs of the service chain are placed based on their appearance on the chains. Figure 6 shows the average *Cost* of the service chains on the configuration with 5, 10, and 15 VNFs per hypervisor, respectively. We can see that for each configuration, the *Cost* of *LightChain* is less than that of *FCFP*. As the number of hypervisors per switch increases, the *Cost* of the chain decreases and converges to one because switches can now accommodate more VNFs on its connected hypervisors.

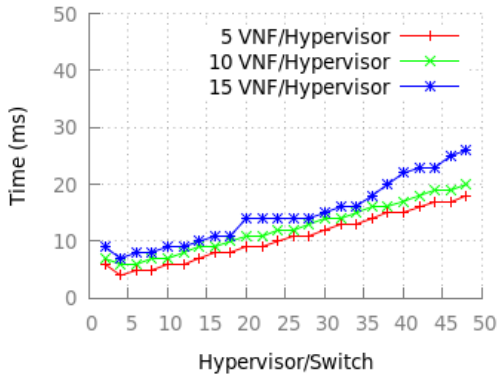

Fig. 5. Average Computation Time to Determine VNFs Placement

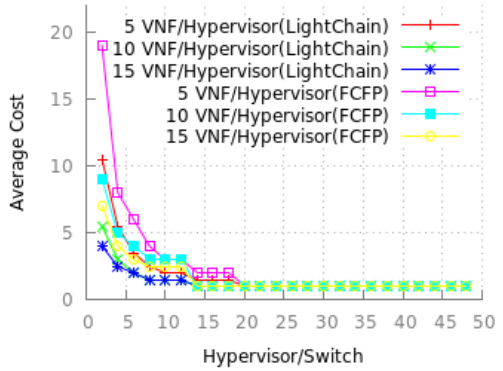

Fig. 6. Average *Cost* of the Service Chain

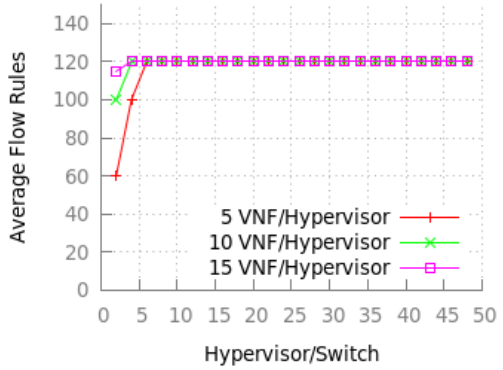

Fig. 7. Average Number of Flow Rules per Switch in LightChain

C. Impact of ping-pong traffic on the number of flow rules

Figure 7 shows the average number of flow rules installed on the switches in the *LightChain*-based approach. As the number

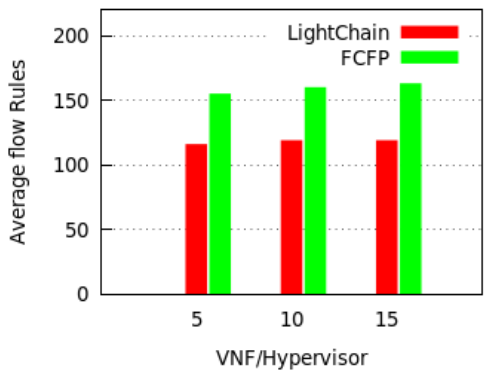

Fig. 8. Average Number of Flow Rules per Switch in LightChain and FCFP

of hypervisor per switch increases, more VNFs can be placed on the hypervisors of the same switch, which results in more flow rules in the switch. A further increase in the hypervisors per switch leads to the convergence of the flow rules because the numbers of hypervisors and switches are large enough to cover the demanded number of service chains.

Figure 8 shows the average number of flow rules per switch in *LightChain* and *FCFP*. We can see that *LightChain* installs fewer flow rules than *FCFP* in all VNFs per Hypervisor configuration. The reason *FCFP* installs more flow rules is because *FCFP* does not consider ping-pong traffic while placing the VNFs. In *LightChain*, traffic for service chains always flows in one direction. Thus, the switches have fewer flow rules than the *FCFP*.

VI. Conclusion and Future Work

This paper addressed the problem of VNF placement, which plays an important role in utilizing the network resources, *i.e.*, switch memory (TCAM) and bandwidth, and has an impact on the performance of services applied to the corresponding flow. *LightChain* uses a heuristic that optimizes the placement of VNFs across service chains in the network in a polynomial run time. Evaluation results show that even for a large number of service chains with different hypervisor per switch configurations, our heuristics run on the order of milliseconds to order the VNFs and determine their placement locations. We also showed that our heuristics reduce the *Cost* of the service chains and number of flow rules in the switches. In the evaluation, the elimination of ping-pong traffic was not evaluated because it is the nature of the proposed algorithm. As future work, we plan to implement *LightChain* in a real NFV environment with larger interconnected data center networks. We will further explore the optimization of our approach on different network topologies, including load balancing, and consider a greater variety of metrics integrated in the algorithm.

References

[1] Network functions virtualisation white paper 3. [Online]. Available: https://portal.etsi.org/Portals/0/TBpages/NFV/Docs/NFV_White_Paper3.pdf

[2] Service function chaining general use cases. [Online]. Available: https://tools.ietf.org/html/draft-liu-sfc-use-cases-08

[3] Service function chaining (sfc) architecture. [Online]. Available: https://tools.ietf.org/html/draft-merged-sfc-architecture-02

[4] Network service chaining problem statementr. [Online]. Available: https://tools.ietf.org/html/draft-quinn-nsc-problem-statement-03

[5] Network service header. [Online]. Available: https://tools.ietf.org/html/draft-quinn-sfc-nsh-07

[6] D. A. Joseph, A. Tavakoli, and I. Stoica, "A policy-aware switching layer for data centers," in *Proceedings of the ACM SIGCOMM 2008 Conference on Data Communication*, ser. SIGCOMM '08. New York, NY, USA: ACM, 2008, pp. 51–62.

[7] B. Anwer, T. Benson, N. Feamster, D. Levin, and J. Rexford, "A slick control plane for network middleboxes," in *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*, ser. HotSDN '13. New York, NY, USA: ACM, 2013, pp. 147–148.

[8] S. K. Fayazbakhsh, L. Chiang, V. Sekar, M. Yu, and J. C. Mogul, "Enforcing network-wide policies in the presence of dynamic middlebox actions using flowtags," in *Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation*, ser. NSDI'14. Berkeley, CA, USA: USENIX Association, 2014, pp. 533–546.

[9] A. Gember, A. Krishnamurthy, S. S. John, R. Grandl, X. Gao, A. Anand, T. Benson, A. Akella, and V. Sekar, "Stratos: A network-aware orchestration layer for middleboxes in the cloud," *CoRR*, vol. abs/1305.0209, 2013.

[10] Z. A. Qazi, C.-C. Tu, L. Chiang, R. Miao, V. Sekar, and M. Yu, "Simple-fying middlebox policy enforcement using sdn," in *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, ser. SIGCOMM '13. New York, NY, USA: ACM, 2013, pp. 27–38.

[11] M. Alicherry and T. Lakshman, "Network aware resource allocation in distributed clouds," in *INFOCOM, 2012 Proceedings IEEE*, March 2012, pp. 963–971.

[12] D. Jayasinghe, C. Pu, T. Eilam, M. Steinder, I. Whally, and E. Snible, "Improving performance and availability of services hosted on iaas clouds with structural constraint-aware virtual machine placement," in *Services Computing (SCC), 2011 IEEE International Conference on*, July 2011, pp. 72–79.

[13] X. Meng, V. Pappas, and L. Zhang, "Improving the scalability of data center networks with traffic-aware virtual machine placement," in *INFOCOM, 2010 Proceedings IEEE*, March 2010, pp. 1–9.

[14] M. Luizelli, L. Bays, L. Buriol, M. Barcellos, and L. Gaspary, "Piecing together the nfv provisioning puzzle: Efficient placement and chaining of virtual network functions," in *Integrated Network Management (IM), 2015 IFIP/IEEE International Symposium on*, May 2015, pp. 98–106.

[15] R. Cohen, L. Lewin-Eytan, J. Naor, and D. Raz, "Near optimal placement of virtual network functions," in *Computer Communications (INFOCOM), 2015 IEEE Conference on*, April 2015, pp. 1346–1354.

[16] J. Sherry, S. Hasan, C. Scott, A. Krishnamurthy, S. Ratnasamy, and V. Sekar, "Making middleboxes someone else's problem: Network processing as a cloud service," in *Proceedings of the ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, ser. SIGCOMM '12. New York, NY, USA: ACM, 2012, pp. 13–24.

[17] L. Li, V. Liaghat, H. Zhao, M. Hajiaghay, D. Li, G. Wilfong, Y. Yang, and C. Guo, "Pace: Policy-aware application cloud embedding," in *INFOCOM, 2013 Proceedings IEEE*, April 2013, pp. 638–646.

[18] A. Kumar, S. Jain, U. Naik, A. Raghuraman, N. Kasinadhuni, E. C. Zermeno, C. S. Gunn, J. Ai, B. Carlin, M. Amarandei-Stavila, M. Robin, A. Siganporia, S. Stuart, and A. Vahdat, "Bwe: Flexible, hierarchical bandwidth allocation for wan distributed computing," in *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, ser. SIGCOMM '15. New York, NY, USA: ACM, 2015, pp. 1–14.

[19] V. Sekar, N. Egi, S. Ratnasamy, M. K. Reiter, and G. Shi, "Design and implementation of a consolidated middlebox architecture," in *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*, ser. NSDI'12. Berkeley, CA, USA: USENIX Association, 2012, pp. 24–24.

[20] T. H. Cormen, C. Stein, R. L. Rivest, and C. E. Leiserson, *Introduction to Algorithms*, 2nd ed. McGraw-Hill Higher Education, 2001.