AggVNF: Aggregate VNF Allocation and Migration in Dynamic Cloud Data Centers

Christopher Gonzalez and Bin Tang
Department of Computer Science
California State University Dominguez Hills, Carson, CA 90747, USA
Email: cgonzalez393@toromail.csudh.edu, btang@csudh.edu

***Abstract*—Service function chaining (SFC), consisting of a sequence of virtual network functions (VNFs), is the de-facto service provisioning mechanism in VNF-enabled data centers (VDCs). However, for the SFC, the dynamic and diverse virtual machine (VM) traffic must traverse a sequence of VNFs possibly installed at different locations at VDCs, resulting in prolonged network delay, redundant network traffic, and large consumption of cloud resources (e.g., bandwidth and energy). Such adverse effects of the SFC, which we refer to as *SFC traffic storm*, significantly impede its efficiency and practical implementation.**

In this paper, we solve the SFC traffic storm problem by proposing *AggVNF*, a framework wherein the VNFs of an SFC are implemented into one aggregate VNF while multiple instances of aggregate VNFs are available in the VDC. AggVNF adaptively allocates and migrates aggregate VNFs to optimize cloud resources in dynamic VDCs while achieving the load balance of VNFs. At the core of the AggVNF are two graph-theoretical problems that have not been adequately studied. We solve both problems by proposing optimal, approximate, and heuristic algorithms. Using real traffic patterns in Facebook data centers, we show that a) our VNF allocation algorithms yield traffic costs 56.3% smaller than the latest research using the SFC design, b) our VNF migration algorithms yield 84.2% less traffic than the latest research using the SFC design, and c) VNF migration is an effective technique in mitigating dynamic traffic in VDCs, reducing the total traffic cost by up to 24.8%.

***Index Terms*—Virtual Network Functions (VNFs), Service Function Chains, Aggregate VNFs, Dynamic Cloud Data Centers**

I. Introduction

Background. Virtual network functions (VNFs), running as virtual machines (VMs) or lightweight containers on commodity hardware, are software implementations of middleboxes (MBs) such as firewalls and TCP optimizers that provide performance and security services in a cloud environment [37]. Consisting of a sequence of VNFs, service function chaining (SFC) has been the primary vehicle to provision multiple VNF services for cloud virtual machine (VM) traffic [7]. Fig. 1(a) shows an SFC with three VNFs. The VM traffic first goes through a firewall VNF, so malicious traffic is filtered, then visits a network load balancer VNF so the traffic congestions can be avoided, and finally passes through a cache proxy VNF so that the network packets can be cached and accessed later. We refer to cloud data centers implementing VNFs as *VNF-enabled data centers (VDCs)*.

As different VNFs in an SFC are usually installed on physical machines (PMs) at various locations in a VDC, steering the massive amount of VM traffic along the SFCs in a large-scale VDC often results in prolonged network delay, incurs an extra amount of network traffic, and consumes considerable cloud resources such as bandwidth and energy [20]. We refer to such intrinsic weakness and adverse effects of SFCs as *SFC traffic storms*. In Fig. 1(a), a single traffic flow has to traverse all six switches to fulfill the SFC requirement, appropriating the resources of the entire network. SFC traffic storms become more severe when deploying SFCs in emerging edge-and-cloud networks [32]. As such networks span much larger geographical areas and edge AI techniques call for more agile network services for processing enormous amounts of data and traffic at the edge [41], how to design and deploy future network function services becomes a pressing challenge.

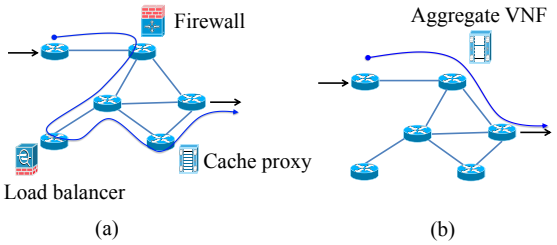


Fig. 1. (a) SFC traffic storm in the VDC. (b) Aggregate VNF in the VDC.

Why SFC? In retrospect, two reasons contributed to the initial adoption of SFCs for cloud service provisions. The first and the historical reason is that early MBs were mainly standalone and proprietary hardware devices providing dedicated services [9]. For cloud operators to offer multiple services to cloud traffic, a natural solution is placing MBs at different strategic points of a network so the traffic can traverse them in some order as a logical chain [23]. The second and technical reason is service function disaggregation (SFD) [18], [13]. As supporting multiple functionalities in a single container or VM requires more compute resources (e.g., CPU cores, memory, and software libraries) and creates resource-heavy VNFs, SFD is proposed to separate the VNF implementation from its underlying hardware to bring down the computational cost and maintain the throughput of the processed traffic flows.

While it is evident that the above historical reason is no longer valid, the arguments for the SFD need some debate. One key observation is that data center servers are generally

underutilized due to over-provisioning for the peak resource demand [43]. For example, a measurement of several thousands of servers randomly selected from different data centers shows the mean utilization of CPUs could be as low as 17.76% [38]. Therefore, although implementing multiple VNFs on the same VM or container is at the expense of computational efficiency, it does not necessarily affect the throughput of the processed cloud traffic.

VNF aggregation is further facilitated by the technical stride of cloud-native network functions (CNFs) [1]. With a small performance footprint and independence of guest OSs, CNFs can containerize micro-services by packing many network functions into a single VM appliance at runtime [17]. As inter-VNF communication latency depends only on inter-process communication when hosted in the same VM, the delay of traversing the SFC is greatly reduced [37].

Our Proposal and Contributions. With all the above investigations, we envision that to solve the SFC traffic storm problem, VNFs of different functions can be implemented in a single CNF located in a single PM. We refer to such a CNF as an *aggregate VNF*. One shortcoming of such a design is the single point of failure and overloading of the same aggregate VNF. This, however, can be overcome by the software nature of VNFs, wherein multiple instances of the same aggregate VNF can be made and installed [11]. With this, each VM traffic flow in the VDC only visits one VNF instance to fulfill the security requirement and performance guarantee. Besides, distributing large amounts of VM network traffic among multiple VNFs achieves load balance and reduces congestion to cloud network traffic. With installing such an aggregate VNF instance in Fig. 1(b), the SFC traffic traverses only a small part of the network, significantly reducing the network traffic and resource consumption. As software-implemented VNFs usually have less processing power than hardware-based MBs, we assume each VNF has a limited processing capacity and can only process a limited amount of VM flows. This is based on the consideration that as aggregating VNFs on a PM (with limited resources) makes the resources distributed for each VNF less, the throughput of the passing data flow will be limited.

We propose a new algorithmic framework called **AggVNF** to allocate and migrate aggregate VNFs in dynamic VDCs adaptively. We first study **ANA**: aggregate VNF allocation in VDCs. Given a set of VM flows with different traffic rates and multiple instances of the aggregate VNFs with various and limited processing capacities, ANA studies how to place the VNFs inside the VDC and then how to assign VM flows to VNFs to minimize the total communication traffic of all VM flows while satisfying the processing capacities of VNFs.[1] Next, due to dynamic cloud traffic in VDCs, such initial VNF allocation may become sub-optimal. We thus propose and study **ANN**: aggregate VNF migration. ANN adaptively responds to the dynamic traffic in the VDC by migrating VNFs around to minimize the total cloud traffic in the VDC.

[1]For ease of presentation, we still refer to aggregate VNFs as VNFs.

We design optimal, approximate, and heuristic VNF allocation and migration algorithms. In particular, for ANA, we propose an integer linear programming (ILP)-based optimal solution, a bi-criteria approximation algorithm based on ILP relaxation, and an efficient greedy algorithm that empirically performs close to the ILP. For ANN, we design an ILP-based optimal solution, a Pareto-optimal solution, and a heuristic algorithm. In particular, our approximation algorithm for ANA achieves a (4, 2) bi-criteria ratio, improving the (6, 4) bi-criteria approximation algorithm by Cohen et al. [14]. Using flow characteristics and real traffic patterns found in Facebook data centers, we show that a) our VNF allocation algorithms yield traffic costs 56.3% smaller than the latest SFC-based VNF placement algorithms [44], b) our VNF migration algorithms yield 84.2% less traffic compared to the latest SFC-based VNF migration algorithms [44], and c) VNF migration is an effective technique in mitigating dynamic traffic in VDCs, reducing the total network traffic by up to 24.8%. Underlying ANA and ANN are two graph-theoretical problems that have not been adequately studied. In particular, ANA generalizes the well-known *capacitated k-median* (CKM) problem [26], [27].

Most of the existing works on SFC placement and migration [22], [46], [31], [16] either proposed ILP solutions (which lack scalability) or heuristic algorithms (which do not have performance guarantees). In contrast, with our aggregate design of VNFs, not only do we solve the traffic storm problem caused by traditional SFC design, but our techniques are time-efficient and have achieved approximation performance for SFC placement and Pareto-optimal for SFC migration, respectively. Therefore, AggVNF and its related techniques are not the same as traditional SFC with a length of one. Furthermore, unlike most existing work, AggVNF can optimize cloud resources for a VDC's lifetime. After the initial aggregate VNF allocation by ANA, the ANN can execute periodically to migrate VNFs to optimize cloud resources in the face of the dynamic VDC traffic.

Paper Organization. Section II provides a background of our work to motivate our contributions. In Section III and IV, we formulate ANA and ANN, respectively, and solve them via various ILP, Pareto-optimal, greedy, and approximation algorithms. In Section V, we conduct extensive experiments for both approaches and discuss their pros and cons. Section VI concludes the paper with some future research.

II. Background

In this section, we first provide an illustrative example, review the related work, and finally present the notations and the network and cost models.

A. An Illustrative Example

Fig. 2 shows a linear VDC with two PMs: p_1 and p_2 and three switches: s_1, s_2, and s_3. There are two communicating VM flows (v_1, v_1') and (v_2, v_2') in the VDC, with v_1 and v_1' located at p_1 and v_2 and v_2' at p_2, and one VNF f_1 to be allocated in the VDC. The two VMs in the same flow

communicate with each other with some traffic rate, which is the frequency or bandwidth demand of this communication. Initially, the traffic rate of (v_1, v_1') is much larger than that of (v_2, v_2'); that is, (v_1, v_1') has heavy traffic while (v_2, v_2') has light traffic. Thus, the best strategy to allocate a VNF into the VDC to achieve minimum network traffic is to place it at s_1, as shown in Fig. 2(a). This way, the heavy traffic of (v_1, v_1') only traverses to s_1 (the dark solid line) while the light traffic of (v_2, v_2') takes a much longer route (the light solid line).

Later on, however, due to the dynamic traffic in the cloud data center, the traffic rate of (v_2, v_2') becomes much larger than that of (v_1, v_1'). As the heavy traffic between v_2 and v_2' (dark solid line in Fig. 2(b)) goes through the entire network and takes a route that is much longer than that of (v_1, v_1'), the network generates more traffic and consumes more bandwidth compared to Fig. 2(a).

One critical observation is that migrating the VNF from one switch to another could mitigate such dynamic network traffic and reduce resource consumption. As shown in the red dash line in Fig. 2(c), we can migrate f_1 from s_1 to s_3. With this VNF migration, the heavy traffic of (v_2, v_2') of traversing f_1 is now restricted to a much smaller part of the network, as shown in Fig. 2(d). Although the light traffic of (v_1, v_1') must traverse much farther to reach f_1, the total network traffic is vastly reduced. Note the processing capacity of f_1 must be at least 2, as two VM flows need to traverse it.

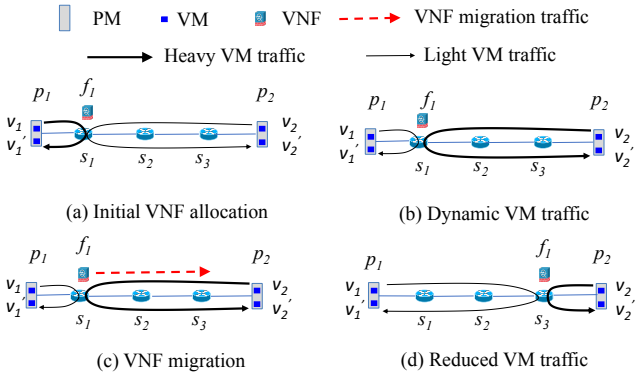


Fig. 2. Illustrating VNF allocation and migration in a linear VDC.

B. Related Work

In recent years, constant research efforts and strides have been made to accelerate and consolidate multiple VNFs into the same VM or container efficiently [17]. Below, we review both the system and algorithmic research in this field.

System Research. CoMb [40] was one of the first to consolidate MBs so that a traffic flow receives all necessary service functions at a single physical machine. Enabled by a shared virtualization platform with a logically centralized network controller, CoMB simplified traffic routing along SFC by reducing the number of routing rules in the switches. Based upon container technologies like Docker [36] and DPDK in Linux containers, openNetVM [45] densely packed NFV appliances with a large number of network functions at runtime and interconnect them to build service chains and NFP [42] proposed a parallelism framework to improve VNF performance and reduce significant latency for SFCs. NFVnice [24], [25] proposed task- and packet-level scheduling that substantially improved the throughput and data packet drop rates. Focusing on the scheduling and control problems of NFs running on shared CPU cores, NFVnice employed backpressure to shed load early in the service chain to prevent wasted work, which complements the capabilities of the OS scheduler. See [17] for a recent review of various techniques for speeding up network function processing.

Algorithm Research. Following the above system progress, a few algorithmic frameworks were proposed to place and route consolidated MBs. Gushchin et al. [19] presented a multipoint-to-point tree for routing traffic in SDN-based networks based on ILP optimization. Huang [21] studied the network throughput maximization problem in an SDN in both snapshot and online scenarios. Recently, Liu et al. [29], [30] considered that MBs could change the volume of processed traffic and focused on the load balance of consolidated MBs. However, none considered the dynamic network environment wherein VNF migration could play a significant role on top of the initial VNF allocation and routing. We propose a comprehensive framework that studies both VNF allocation and migration for aggregate VNFs in a dynamic cloud environment.

Cohen et al. [14] studied the static VNF placement problem, and proposed a (6, 4) bi-criteria approximation algorithm (i.e., it achieves at most six times the optimal cost while using at most four times the allowed node resource). It considered VNF setup costs, connectivity costs of the clients, and capacity constraints of the network nodes and modeled it as a combination of *facility location problem* and *generalized assignment problem*. In contrast, our approximation algorithm for ANA, while without setup cost, achieves a better (4, 2) bi-criteria ratio by utilizing Markov's inequality [5]. Besides, they only studied the static VNF allocation problem and did not consider VNF migration with dynamic cloud network traffic. Recently, Tran et al. [44] proposed a primal-dual-based $2+\epsilon$ approximation algorithm for traffic-optimal SFC placement, and their network setup is close to ours. We thus compare our algorithms with theirs.

C. Notations and Preliminaries

Network Model. We model a VDC as an undirected graph $G(V, E)$; $V = V_p \cup V_s$ includes a set V_p of PMs and a set V_s of switches and E is the set of edges. We use fat-trees [8] to illustrate the problems and for simulations, which, however, are applicable to any data center topology. As more than half of a data center traffic is pairwise east-west cloud traffic [2], [35], we focus on pairwise VM communication and assume there are $|L|$ communicating VM flows $L = \{l_1, l_2, ..., l_{|L|}\}$. For the flow $l_i = (v_i, v_i')$, v_i and v_i' communicate with each other following their traffic rate, which indicates their

communication frequency or bandwidth demand. Each VM $v \in \mathcal{V} = \{v_1, v_1', ..., v_{|L|}, v_{|L|'}\}$ is located at a PM $s(v) \in V_p$.

Let λ_i denote the *initial traffic rate* of flow l_i and $\lambda = \langle \lambda_1, \lambda_2, ..., \lambda_{|L|} \rangle$ the *initial traffic rate vector* of the $|L|$ flows. In a dynamic VDC, the traffic rates of VM flows change over time; thus, λ is not a constant.

VNF Model. There are $|F|$ aggregate VNF instances $F = \{f_1, f_2, ..., f_{|F|}\}$ to be allocated into the VDC. We assume each switch is attached to a server that can install VNFs. As aggregate VNFs usually consume more computational resources than individual VNFs, we assume the aggregation VNFs are installed on servers on different switches [46]. That is, if f_j is installed on switch $p(j)$ and $f_{j'}$ on switch $p(j')$ and $j \neq j'$, then $p(j) \neq p(j')$. The *processing capacity* of f_j, $1 \leq j \leq |F|$, is κ_j, meaning at most κ_j VM flows can traverse f_j at the same time (we leave the case that processing capacity depends on the traffic rates of individual flows as future work). For security and performance purposes, each communicating VM flow l_i must traverse one of the VNF instances; therefore $\sum_{j=1}^{|F|} \kappa_j \geq |L|$.

Cost Model. Each edge $(u, v) \in E$ has a cost $w_{u,v}$, indicating the delay or energy cost on this edge for one unit of VM communication or VNF migration. Given any PM or switch u and v, let $c(u, v)$ denote the shortest path cost between u to v. We define the *VM communication cost* of any flow l_i as $\lambda_i \cdot c\big(s(v_i), s(v_i')\big)$ and the *VNF migration cost* of migrating any VNF from one switch u to another switch v as $\mu \cdot c\big(u, v\big)$. Here μ is *VNF migration coefficient*, which is the ratio between the costs of VNF migration and VM communication. μ represents the relative size of memory or data packet transferred (thus delay) in VNF migration and VM communication.

III. ANA: Aggregate VNF Allocation

ANA consists of two sequential stages: *VNF placement* (i.e., which VNF is placed on which switch) and *VM flow assignment* (i.e., which VM flow traverses which VNF).

A. Problem Formulation

The *VNF placement function* $p : F \to V_s$ places VNF $f_j \in F$ at switch $p(j) \in V_s$ and then *VM flow assignment function* $a : L \to F$ assigns VM flow $l_i \in L$ to traverse $f_{a(i)} \in F$. Given the initial traffic rate vector λ, a VNF placement p, and a VM flow assignment a, the *total communication cost* of all the l VM flows $C_c(\lambda, p, a) == \sum_{i=1}^{|L|} c_{i,p(a(i))} = \sum_{i=1}^{|L|} \lambda_i \cdot \Big(c\big(s(v_i), p(a(i))\big) + c\big(p(a(i)), s(v_i')\big)\Big)$. Here, $c_{i,k}$ indicates the communication cost of l_i when traversing the VNF placed at switch k. The objective of ANA is to find a p and an a to minimize $C_c(\lambda, p, a)$ while satisfying capacity constraint of VNFs: $|\{a(i) = j, 1 \leq i \leq |L|\}| \leq \kappa_j, 1 \leq j \leq |F|$.

Theorem 1: The ANA is NP-hard.

Proof: Here, we only give a proof sketch due to space constraints. We show that *uniform capacitated k-median placement problem* (uCKM) [27], which is NP-hard, is equivalent to a special case of ANA with $\kappa_j = \kappa$; that is, when all the VNFs have the same processing capacity. uCKM is defined as follows. Given a set F of facilities, each facility $f \in F$ has the same uniform capacity $u \geq 0$, a set C of clients, a metric d over $F \cup C$, and an integer k. The goal of uCKM is to find a set $S \subset F$ of at most k open facilities and a connection assignment $\phi : C \to S$ of clients to the open facilities to minimize the connection cost $\sum_{c \in C} d(c, \phi(c))$ while satisfying that $|\phi^{-1}(f)| \leq u$. ∎

Note that ANA generalizes the well-known *capacitated k-median* (CKM) problem [26], [27]. CKM opens k facilities (each with a fixed capacity) to minimize the total access cost of demand nodes. In contrast, by placing VNFs of different capacities on switches (thus "opening facilities" for VM flow access), ANA can assign different capacities to the facilities.

B. Integer Linear Programming Solution for ANA

We solve ANA optimally by formulating it as an integer program ILP(A) below. There are two decision variables: $x_{j,k}$ indicates if VNF f_j is placed on switch k and $y_{i,k}$ indicates if VM flow (v_i, v_i') traverses VNF placed at switch k.

$$(A) \quad \min \sum_{i=1}^{|L|} \lambda_i \sum_{k=1}^{|V_s|} (c_{i,k} \cdot y_{i,k}) \tag{1}$$

s.t.

$$x_{j,k} \in \{0, 1\} \quad \forall 1 \leq j \leq |F|, 1 \leq k \leq |V_s| \tag{2}$$

$$y_{i,k} \in \{0, 1\} \quad \forall 1 \leq i \leq |L|, 1 \leq k \leq |V_s| \tag{3}$$

$$\sum_{k=1}^{|V_s|} x_{j,k} = 1, \quad \forall 1 \leq j \leq |F| \tag{4}$$

$$\sum_{j=1}^{|F|} x_{j,k} \leq 1, \quad \forall 1 \leq k \leq |V_s| \tag{5}$$

$$y_{i,k} \leq \sum_{j=1}^{|F|} x_{j,k}, \quad \forall 1 \leq i \leq |L|, 1 \leq k \leq |V_s| \tag{6}$$

$$\sum_{k=1}^{|V_s|} y_{i,k} = 1, \quad \forall 1 \leq i \leq |L| \tag{7}$$

$$\sum_{i=1}^{|L|} y_{i,k} \leq \sum_{j=1}^{|F|} (x_{j,k} \cdot \kappa_j), \quad \forall 1 \leq k \leq |V_s| \tag{8}$$

Objective function 1 is to minimize the VM flows' total communication cost. Equations 2 and 3 are the integer constraints of $x_{j,k}$ and $y_{i,k}$, respectively. Equation 4 guarantees that each of the $|F|$ VNFs must be placed on some switch. Inequality 5 indicates that each switch is placed with at most one VNF. Inequality 6 guarantees that if a VM flow traverses a switch, there must be a VNF installed on that switch. Equation 7 ensures each VM flow must traverse one switch, and Inequality 8 enforces VNF processing capacity constraint.

C. Heuristic Algorithm for ANA

As computing ILP(A) is time-consuming for large-scale VDCs, we present a time-efficient greedy heuristic algorithm viz. Algo. 1 below. Its main idea is to allocate VNFs with high

processing capacity earlier to accommodate VM flows with large traffic rates; this way, heavy-traffic VM communication can choose closer VNFs at the price of light traffic going to farther VNFs, thus reducing network traffic. First, it sorts the VNFs and VM flows in the non-ascending order of their capacities and traffic rates, respectively (lines 1-2). Then, it finds a switch to place VNF f_j such that this placement minimizes the total communication cost of the next κ_j VM flows assigned to f_j (lines 3-20). This continues until all the VNFs are placed, each VM flow is assigned to one VNF, and it returns the total communication cost of all the VM flows (lines 21-22). The time complexity of Algo. 1 is $O(|F| \cdot \log|F| + |L| \cdot \log|L| + |F| \cdot |V_s| \cdot |L|)$.

Algorithm 1: Greedy Algorithm for ANA.
Input: A VDC, VM placement $s(v)$, and traffic rate vector λ;
Output: VNF placement function p, VM flow assignment function a, and total communication cost $C_c(\lambda, p, a)$.
Notations: $index$: index of VM flow assigned to next VNF; $occupied_k$: if switch k is installed a VNF, initially false;
1. Sort f_j, $1 \le j \le |F|$, in the non-ascending order of their capacities κ_j. Let $\kappa_1 \ge \kappa_2 \ge ... \ge \kappa_{|F|}$;
2. Sort l_i, $1 \le i \le |L|$, in the non-ascending order of their traffic rates λ_i. Let $\lambda_1 \ge \lambda_2 \ge ... \ge \lambda_{|L|}$;
3. $index = 1$; // VM flow with largest traffic rate
4. **for** ($j = 1$ to $|F|$)
5. $\tau = +\infty$, $sum = 0$;
6. **for** ($k = 1$ to $|V_s|$)
7. **if** ($occupied_k == false$)
8. **for** ($i = index$ to $(index + \kappa_j - 1)$)
9. $sum = \lambda_i \cdot \big(c(s(i), k) + c(k, s(i'))\big)$;
10. **if** ($sum < \tau$)
11. $\tau = sum$, $p(j) = k$;
12. **end for;**
13. **end if;**
14. **end for;**
15. $occupied_{p(j)} = true$;
16. **for** $\big(i = index$ to $(index + \kappa_j - 1)\big)$
17. $a(i) = p(j)$;
18. **end for;**
19. $index = index + \kappa_j$;
20. **end for;**
21. Compute $C_c(\lambda, p, a)$;
22. **RETURN** p, a, and $C_c(\lambda, p, a)$.

D. Approximation Algorithm for ANA

Next, we propose a bi-criteria approximation algorithm Algo. 2 for ANA when $\kappa_j = \infty$, $1 \le j \le |F|$. It is inspired by Lin and Vitter [28] and following work [3] and is based on the relaxation of ILP(A), wherein Equations 2 and 3 become $0 \le x_{j,k} \le 1$ and $0 \le y_{i,k} \le 1$, respectively. Let $C_i = \sum_{k=1}^{|V_s|}(c_{i,k} \cdot y_{i,k})$, where $y_{i,k}$ is computed from the ILP relaxation. That is, C_i is the expected distance from VM flow l_i to all the VNFs it accesses. Denote the total cost of the VM flows with the above ILP relaxation as t^{LP}; $t^{LP} = \sum_{i=1}^{|L|} C_i$.

Definition 1: (**Covered-Neighbors of Node** i.) We define *2-neighbors* of node i in a VDC graph as the set of nodes whose distances to i is at most $2 \cdot C_i$ and denote it as $N(i)$; that is, $N(i) = \{j \in V | c(i, j) \le 2 \cdot C_i\}$. Define the *covered-neighbors* of i as the set of nodes whose 2-neighbors and i's 2-neighbors have at least one common node, and denote it as $C(i)$; that is, $C(i) = \{j \in V | N(i) \cup N(j) \ne \emptyset\}$. We have $\{i\} \subseteq N(i) \subseteq C(i)$. □

Algo. 2 works as follows. First, it computes the relaxation of the ILP(A) to find C_i and initializes S to be V (lines 1-2). Then it takes place in rounds; in each round, it selects a node $i \in S$ with the minimum C_i to place a VNF and removes all nodes in $C(i)$ from S (lines 3-7). This continues until all the nodes are removed from S. After that, it assigns each VM flow to the closest VNF and computes the cost accordingly (lines 8-12). Its running time is dominated by the ILP relaxation, which is $O\big((|L| + |F|) \cdot |V_s|\big)$ [34].

Algorithm 2: Approximation Algorithm for ANA.
Input: A VDC $G(V, E)$ with VM flow placement $s(v)$, and initial traffic rate vector λ;
Output: VNF placement function p, VM flow assignment function a, and total communication cost $C_c(\lambda, p, a)$.
Notations: S: the set of nodes, initially $S = V$; p: the set of nodes selected to place VNFs, initially $p = \emptyset$.
1. Compute ILP(A) relaxation, $C_i = \sum_{k=1}^{|V_s|}(c_{i,k} \cdot y_{i,k})$;
2. $S = V$, $p = \phi$;
3. **while** ($S \ne \emptyset$)
4. Let $s = \text{argmin}_{i \in S} C_i$;
5. $p = p \cup \{s\}$;
6. $S = S - C(i)$;
7. **end while;**
8. $C_c(\lambda, p, a) = 0$;
9. **for** ($i = 1$ to $|L|$) // VM flow assignment and total cost
10. $a(i) = \text{argmin}_{k \in p}(c(s(i), k) + c(k, s(i')))$;
11. $C_c(\lambda, p, a) += \lambda_i \cdot (c(s(i), a(i)) + c(a(i), s(i')))$;
12. **end for;**
13. **RETURN** p, a, and $C_c(\lambda, p, a)$.

Below we prove that Algo. 2 is a bi-criteria approximation algorithm with $|p| \le 2 \cdot |F|$ and $C_c(\lambda, p, a) \le 4 \cdot t^{OPT}$, where t^{OPT} is the optimal cost computed by ILP(A).

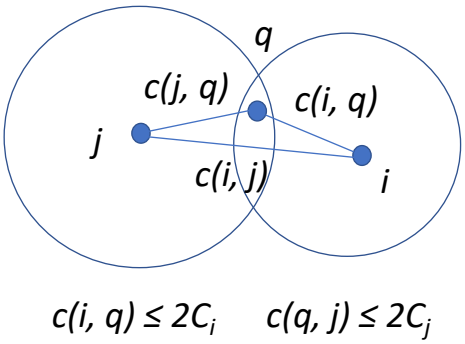


Fig. 3. Illustrating the approximation ratio of Algo. 2.

Lemma 1: At any round in Algo. 2, if node i is selected to place a VNF, for any $j \in C(i) \subseteq S$, we have $c(i,j) \leq 4 \cdot C_j$.

Proof: As $j \in C(i)$, then $N(i) \cap N(j) \neq \emptyset$ according to its definition; let $Q = N(i) \cap N(j)$. Given any node $q \in Q$, we have $c(i,j) \leq c(i,q) + c(q,j) \leq 2C_i + 2C_j$, as shown in Fig. 3. As both i and j are available in S when i is selected, $C_i \leq C_j$. Therefore $c(i,j) \leq 4C_j$. ∎

Theorem 2: $C_c(\lambda, p, a) \leq 4 \cdot t^{OPT}$ and $|p| \leq 2 \cdot |F|$.

Proof: In Algo. 2, let the cost of each VM flow l_i accessing its closest VNF as c_i; that is, $C_c(\lambda, p, a) = \sum_{l_i \in L} c_i$. For any node $q \in S$, as $q \in C(i)$ for some $i \in p$, we have $c_q \leq 4 \cdot C_i \leq 4 \cdot C_q$ (Lemma 1). So $C_c(\lambda, p, a) = \sum_{l_i \in L} c_i = \sum_{q \in S} c_q \leq 4 \cdot \sum_{q \in S} C_q = 4 \cdot t^{LP} \leq 4 \cdot t^{OPT}$. Recall t^{LP} is the total cost of the relaxation of ILP(A).

To prove $|p| \leq 2 \cdot |F|$, as $\sum_k \sum_{j=1}^{|F|} x_{j,k} \leq |F|$ and $y_{i,k} \leq \sum_{j=1}^{|F|} x_{j,k}$ (i.e., Inequality 6), we show that $\sum_{j \in N(i)} x_{j,k} \geq \frac{1}{2}$. We define a random variable Z that takes $c_{i,k}$ with probability $y_{i,k}$. The expected value of Z is $\mathbb{E} = \sum_{k=1}^{|V_s|} (c_{i,k} \cdot y_{i,k}) = C_i$. Using Markov's inequality [5], $\sum_{j \in C(i)} y_{i,k} = \mathbb{P}[Z \leq 2 \cdot C_i] = 1 - \mathbb{P}[Z > 2 \cdot \mathbb{E}Z] \geq \frac{1}{2}$. ∎

IV. ANN: Aggregate VNF Migration

Due to the dynamic traffic in a VDC, the VM flows' new traffic rate vector now becomes $\lambda' = \langle \lambda'_1, ..., \lambda'_{|L|} \rangle$. As such, the VNF placement p and VM flow assignment a computed in Section III may not be optimal; i.e., $C_c(\lambda, p, a) \leq C_c(\lambda', p, a)$. This necessitates aggregate VNF migration, viz. ANN.

A. Problem Formulation of ANN.

We define *VNF migration function* $p' : F \rightarrow V_s$, which migrates VNF $f_j \in F$ from its current switch $p(j) \in V_s$ to another switch $p'(j) \in V_s$ ($p(j) = p'(j)$ means f_j does not migrate) and *VM flow assignment function* $a' : L \rightarrow F$, which assigns VM flow $l_i \in L$ to traverse $f_{a'(i)} \in F$.

The *total migration cost* of all the VNFs is $C_m(p') = \mu \cdot \sum_{j=1}^{|F|} c\big(p(j), p'(j)\big)$. Let $C_c(\lambda', p', a')$ be the *total communication cost* of all VM flows *after* VNF migration scheme p' is done. Let $C_t(\lambda', p', a') = C_m(p') + C_c(\lambda', p', a')$ be the *total cost* of VNF migration and VM communication.

$$\begin{aligned} & C_t(\lambda', p', a') \\ &= \mu \cdot \sum_{j=1}^{|F|} c\big(p(j), p'(j)\big) + \\ & \lambda'_i \cdot \sum_{i=1}^{|L|} \Big(c\big(s(v_i), p'(a'(i))\big) + c\big(p'(a'(i)), s(v'_i)\big) \Big). \end{aligned} \tag{9}$$

The objective of ANN is to find a VNF migration p' and a new VM flow assignment a' such that $C_t(\lambda', p', a')$ is minimized under the capacity constraint of VNFs: $|\{a'(i) = j, 1 \leq i \leq |L|\}| \leq \kappa_j, 1 \leq j \leq |F|$. When $\mu = 0$, ANN degenerates to ANA; thus, ANN is also NP-hard.

B. Algorithms for ANN.

1) ILP Solution: ANN can be solved using the integer program ILP(B) below. Decision variable $x_{j,k}$ indicates if VNF f_j is migrated to switch k and $y_{i,k}$ indicates if VM flow l_i traverses VNF placed at switch k. Recall $c(p(j), k)$ is the migration cost of f_j migrating from switch $p(j)$ to switch k while $c_{i,k}$ is the communication cost of VM pair l_i traversing to the VNF located at switch k.

The objective function 10 in ILP(B) is to minimize the sum of the total migration cost of all the VNFs (first term) and the total communication cost of all the $|L|$ VM flows after the migration (second term). The constraints for ILP(B) are the same as those in ILP(A), except that the switches here refer to where VNFs migrate. We thus omit the constraints.

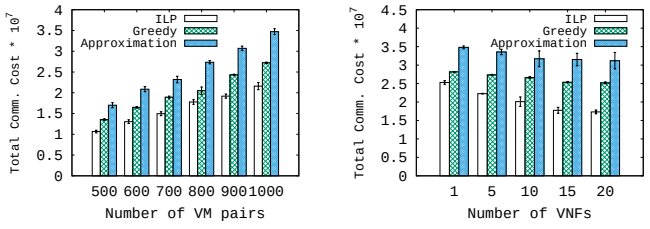
$$\begin{aligned} (B) \qquad \min\Big(& \mu \cdot \sum_{j=1}^{|F|} \sum_{k=1}^{|V_s|} \big(c(p(j), k) \cdot x_{j,k} \big) \\ & + \sum_{i=1}^{|L|} \lambda'_i \sum_{k=1}^{|V_s|} (c_{i,k} \cdot y_{i,k}) \Big) \end{aligned} \tag{10}$$

2) VNF Migration Heuristic Algorithm: Next, we present Algo. 3, a more time-efficient VNF migration heuristic algorithm. Given a new traffic rate vector λ', we first recompute a new VNF placement g and a new VM flow assignment a' using Algo. 1 or 2. We have $C_c(\lambda', g, a') \leq C_c(\lambda', p, a)$. To reduce VM communication cost, a good strategy is to migrate VNF f_j from their current location $p(j)$ towards its new location $g(j)$ along its shortest path while having l_i traversing its newly assigned VNF $f_{a'(i)}$ accordingly. However, as such VNF migration incurs traffic costs, the challenge is to decide how far each VNF migrates. We first give the below definition.

Definition 2: (**VNF Migration Lines.**) Denote the shortest path between $p(j)$ and $g(j)$ as S_j. The *VNF migration lines*, denoted as $\mathcal{F}$, are sets of $|F|$ switches, each from a different S_j. I.e., $\mathcal{F} = \{\{s_{k_1}, s_{k_2}, ..., s_{k_{|F|}}\} | s_{k_j} \in S_j, 1 \leq j \leq |F|\}$. □

Let $|S_j|$ be the number of switches on S_j ($|S_j| = 1$ if $p(j) = g(j)$). There are *at most* $\prod_{j=1}^{|F|} |S_j|$ VNF migration lines in $\mathcal{F}$, each representing one intermediate VNF migration stage when migrating VNFs from the old VNF placement p to the new placement g. Algo. 3 below is thus to find a VNF migration line p' that gives the minimum total cost of VNF migration and VM communication. Algo. 3 takes $O(|V_s|^{|F|})$.

Algorithm 3: VNF Migration Algorithm for ANN.
Input: A VDC $G(V, E)$ with VNF placement $p(j)$ and VM flow assignment $a(i)$, μ, new traffic rate vector λ';
Output: A VNF migration p', new VM flow assignment a', and the total cost $C_t(\lambda', p', a')$.
1. Compute new VNF placement g and new VM flow assignment a' using Algo. 1 or Algo. 2;
2. Among at most $|\mathcal{F}|$ VNF migration schemes, find one giving minimum $C_t(\lambda', p', a')$ and denote it as p';
3. Migrates each f_j from switch $p(j)$ to switch $p'(j)$;
4. **RETURN** p', a', and $C_t(\lambda', p', a')$.



(a) Varying $|L|$, $|F| = 7$. (b) Varying $|F|$, $|L| = 1000$.

Fig. 4. Comparing VNF allocation algorithms.

Pareto Front of VNF Migration. To minimize the total cost, Algo. 3 strikes a balance between VNF migration cost $C_m(p')$ and VM communication cost $C_c(\lambda', p', a')$. It endeavors to find *Pareto-optimal* point, which is a state where neither $C_m(p')$ nor $C_c(\lambda', p', a')$ can be improved without deteriorating the other. Thus ANN is similar to a multi-objective optimization problem (MOOP) [12] that minimizes multiple objectives simultaneously. As Pareto front [12] consisting of Pareto-optimal points is an "optimal" solution of MOOPs, and we can demonstrate that as $C_m(p')$ cannot be reduced without increasing $C_c(\lambda', p', a')$, we argue that all the VNF migration lines yield a Pareto front. Below we give a sufficient condition for the optimality of Algo. 3 using scalarization [12], a popular technique that linearly combines multiple objective functions into a single objective function. It serves as a tool to examine the existence of efficient solutions for MOOPs. Eq. 9 in Sec. IV-A is indeed a scalarization of $C_m(p')$ and $C_c(\lambda', p', a')$. It is well-known that if the solution function of a MOOP has a convex Pareto front, scalarization can identify optimal solutions [15]. We give below theorem without proof.

Theorem 3: If the Pareto front of VNF migration generated by Algo. 3 is convex, Algo. 3 gives the minimum total cost of VM communication and VNF migration.

The above techniques were inspired by Tran et al. [44] that solve a related SFC migration problem. The VNF migration line in [44] is an SFC that VM traffic must traverse in a specific order, whereas in this work, each VM flow traffic only reaches one VNF in the migration line. As such, Algo. 3 can be run periodically in response to dynamic traffic in VDCs.

V. Performance Evaluation

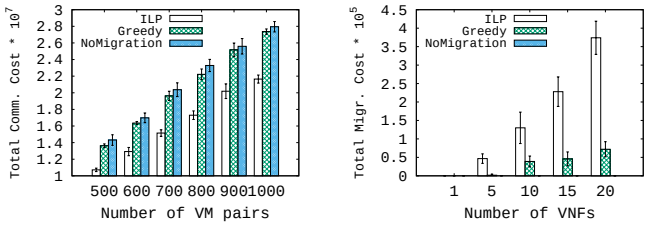
Simulation Setup. We investigate the performance of our algorithms using k=8 fat-tree VDCs with 128 PMs. ILP(A) and (B) are computed using lp_solve [4]. For all other algorithms, we write our simulator in Java. All experiments are performed on a Linux workstation (Ubuntu 20.04 LTS) with an Intel Core processor and 64GB of memory. Each data point in the plots averages 20 runs with a 95% confidence interval. As μ represents the relative costs between VNF migration and VM communication, we quantify it as the ratio between VM communication packet size and memory transferred in VNF migration. As a typical network packet is around 1KB and the size of transferred memory in migrating a containerized VNF is about 100MB [33], we set μ between 10^4 and 10^5.

Real-world Traffic Pattern. Measurements of flow characteristics in Facebook data centers [39] show that 70% of flows send less than 10 KB and last less than 10 seconds, the median flow sends less than 1 KB and lasts less than a second, and less than 5% of the flows are larger than 1 MB or last longer than 100 seconds. To emulate such a traffic pattern, we set the traffic rates of VM flows in the range of [0, 10000], where 25% of VM flows with *light traffic rates* in [0, 3000), 70% *medium traffic rates* in [3000, 7000], and 5% *heavy traffic rates* in (7000,10000]. Besides, as a previous study showed that 80% of cloud east-west cloud traffic stays within the rack [10], we place 80% of the VM flows into PMs under the same edge switches. For each comparison, we set the VNF processing capacity κ_j as a random number in [1, $\lceil \frac{2|L|}{|F|} \rceil$], where $|L|$ is the number of VM flows, and $|F|$ is the number of aggregate VNFs. This is the most stressful scenario, as each VNF operates at its maximum processing capacity. As a typical real-world SFC [6] has at most 10 functions, we consider up to 20 VNFs in our experiments.

State-of-the-Art. Tran et al. [44] studied the traffic-optimal SFC placement and migration that minimize the total network traffic in VDCs. For SFC placement, they proposed a primal-dual-based $2 + \epsilon$ approximation algorithm and a dynamic programming (DP)-based algorithm. For SFC migration, they offered a Pareto-optimal algorithm. As it was shown that their algorithms outperform other existing SFC placement and migration techniques, we compare with their algorithms. We refer to their DP-based SFC placement algorithm and Pareto-optimal SFC migration algorithm as **SFC**.

Algorithms for ANA. For the VNF allocation algorithms, we compare ILP-based optimal solutions viz. ILP(A) (referred to as **ILP**), greedy algorithm viz. Algo. 1 (referred to as **Greedy**), and approximation algorithm Algo. 2 (referred to as **Approximation**). Fig. 4(a) compares ILP, Greedy, and Approximation by varying $|L|$ while fixing $|F|$. It shows that ILP outperforms Greedy, which outperforms Approximation. Although the Approximation has a provable theoretical guarantee, it does not perform as well as the Greedy empirically. Fig. 4(b) varies $|F|$ while fixing $|L|$. Unlike in Fig. 4(a), the network cost of Greedy stays the same with the increase in $|F|$. This is because VNF's processing capacity $\kappa_j = \lceil \frac{2|L|}{|F|} \rceil$ decreases with the increase of $|F|$ when it operates at its full capacity. For ILP and Approximation, its cost slightly decreases with the increase of $|F|$. This is because some VNFs can be placed closer to some VM flows with higher cost (recall that 80% of VM flows are under the same edge switch), thus lowering the total cost. The above observations show that our aggregate VNF design is scalable.

Algorithms for ANN. Next, we create a dynamic traffic scenario and show the traffic-mitigation effect of VNF migration. To generate the dynamic traffic and mimic the Facebook flow pattern simultaneously, at the beginning of each epoch, we



(a) Varying $|L|$, $|F| = 7$. (b) Varying $|F|$, $|L| = 1000$.

Fig. 5. Comparing VNF migration algorithms. $\mu = 10^4$.

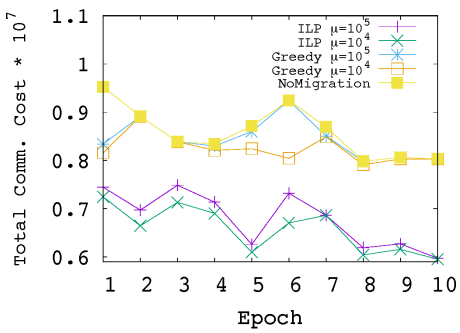
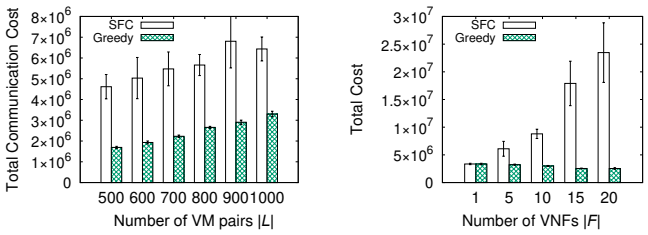


Fig. 6. Dynamic VNF Migration in 10 epochs. $|L| = 300$, $|F| = 7$.



(a) Varying $|L|$, $|F| = 7$. (b) Varying $|F|$, $|L| = 1000$.

Fig. 7. Comparing aggregate VNFs with SFC in (a) VNF allocation and (b) VNF migration.

randomly select three Pods out of the eight Pods in the $k = 8$ fat-tree, and set the VM flows under each as having light, medium, and heavy traffic rates, respectively.

We compare ILP-based optimal solutions viz. ILP(B) (referred to as **ILP**), greedy algorithm viz. Algo. 3 (referred to as **Greedy**), and the case that no VNF migration takes place (referred to as **NoMigration**). Fig. 5(a) shows that both Greedy and ILP yield fewer total communication costs of VM flows than NoMigration, demonstrating that VNF migration effectively reduces dynamic cloud network traffic. In particular, ILP reduces the total communication cost by 24.8% upon NoMigration. Fig. 5(b) shows that Greedy's VNF migration cost is about 18.2% of that of ILP while NoMigration costs zero. As ILP is the optimal VNF migration scheme, it can cost more on migration to minimize the total cost of VM communication and VNF migration. Comparing Fig. 5(a) and (b), the VNF migration cost is around 1% of the VM communication cost. This shows that the migration cost overhead is insignificant in the total network traffic cost, demonstrating the efficacy of our VNF migration algorithms.

Dynamic VDC Traffic. Next, we investigate the performance of our VNF migration solutions. Fig. 6 shows the performances of ILP and Greedy for ten epochs while varying the values of μ as 10^4 and 10^5. At the beginning of each epoch, we randomly select two, five, and one Pod from the fat-tree and set the VM flows under each as having light, medium, and heavy traffic rates, respectively. It then executes ILP and Greedy and calculates the total migration and communication cost. For NoMigration, it simply recalculates the total communication cost using the new traffic rates. It shows that ILP migration can reduce the traffic in all the epochs while Greedy can in five epochs. As Greedy is a "myopic" algorithm, sometimes it treats the no migration as local minimal, thus performing the same as NoMigration. For both Greedy and ILP, the costs of $\mu = 10^5$ are higher than that of $\mu = 10^4$, showing that as the value of μ increases, the total traffic costs increase.

Comparing with SFC. Next, we compare the network traffic costs of the aggregate VNF design with the latest work of SFC design [44], wherein each VM flow must traverse all the $|F|$ VNFs in a sequence (referred to as **SFC**). Fig. 7(a) compares SFC with our Greedy VNF allocation algorithm viz. Algo. 1 by varying VM flows $|L|$. It shows Greedy outperforms SFC by more than 56.3% in terms of the total communication cost of VM flows in the entire range of $|L|$, clear evidence of our aggregate VNF design in reducing the SFC traffic storm. Fig. 7(b) compares SFC with our Greedy VNF migration algorithm viz. Algo. 3 in terms of total VM communication and VNF migration cost by varying the number of VNFs $|F|$. When $|F|$ is one, Greedy and SFC yield the same total cost as all VM flows must go through only VNF. However, when $|F| \geq 2$, Greedy outperforms SFC by yielding less total network cost. The performance difference increases dramatically with the increase of $|F|$. With 20 VNFs in the SFC, Greedy yields 84.2% of less network traffic. This demonstrates the efficacy of the aggregate VNF design again to overcome the traffic storms caused by the SFC design of VNFs. Note that, unlike SFC, increasing $|F|$ does not increase the network cost for Greedy, due to that $\kappa_j = \lceil \frac{2|L|}{|F|} \rceil$.

VI. Conclusions and Future Work

We proposed aggregate VNFs to alleviate the traffic storms incurred by traditional SFC design and establish an algorithmic framework for aggregate VNFs. We formulated two new graph-theoretical aggregate VNF allocation and migration problems and proposed ILP-optimal, Pareto-optimal, approximate, and heuristic algorithms to solve them. Using real traffic patterns in production data centers and real data for VNF resource consumption, we show our aggregate VNF design outperforms the latest SFC-based design by a large margin while achieving high throughput of traffic flow in most cases.

Whereas NFVs and SFCs were developed for the purpose of service function disaggregation, we propose to return back to the root of monolithic design to reduce the delay and traffic in

SFCs. This is critical considering that SFCs are increasingly deployed in edge clouds spanning large geographical areas, wherein edge AI techniques call for more robust and agile services for enormous amounts of data and traffic available at the edge [41]. Consequently, SFC traffic storms could get worse. Our goal, therefore, is to shed light and stir discussion to rethink the traditional SFC design to benefit future networks, or at least strike a balance between these two approaches.

For future work, as AggVNF is a new design concept not being adequately explored, we will conduct a more in-depth and extensive study of their computational efficiency and resource consumption. We will augment our AggVNF model by considering that each switch can store multiple aggregate VNFs and different VM traffic could request different types of aggregate VNFs. Finally, we will consider that the processing capacity of a VNF is dependent on the traffic rates of flows. How to allocate and migrate aggregate VNFs in these more challenging scenarios to adaptively respond to dynamic cloud and edge traffic becomes a suite of new problems.

ACKNOWLEDGMENT

This work was supported by NSF Grant CNS-1911191.

REFERENCES

[1] Cisco cloud-native network functions. https://www.cisco.com/c/en/us/solutions/service-provider/industry/cable/cloud-native-network-functions.html.

[2] Cisco global cloud index: Forecast and methodology, 2016 to 2021 white paper. https://www.cisco.com/c/en/us/solutions/service-provider/global-cloud-index-gci/white-paper-listing.html.

[3] The k-median clustering problem. https://cseweb.ucsd.edu/d̃asgupta/291-geom/kmedian.pdf.

[4] Linear programming solver lp_solve. https://sourceforge.net/projects/lpsolve/.

[5] Markov's inequality. https://en.wikipedia.org/wiki/Markov's_inequality.

[6] Service function chaining use cases in data centers (ietf). https://tools.ietf.org/html/draft-ietf-sfc-dc-use-cases-06section-3.3.1.

[7] A survey on service function chaining. *Journal of Network and Computer Applications*, 75:138–155, 2016.

[8] M. Al-Fares, A. Loukissas, and A. Vahdat. A scalable, commodity data center network architecture. *SIGCOMM Comput. Commun. Rev.*, 38(4):63–74, 2008.

[9] M. Arregoces and M. Portolani. *Data Center Fundamentals*. Cisco Press, 2003.

[10] T. Benson, A. Akella, and D. A. Maltz. Network traffic characteristics of data centers in the wild. In *ACM IMC 2010*.

[11] F. Carpio, S. Dhahri, and A. Jukan. Vnf placement with replication for load balancing in nfv networks. In *IEEE ICC 2017*.

[12] J. Cho, Y. Wang, I. Chen, K. S. Chan, and A. Swami. A survey on modeling and optimizing multi-objective systems. *IEEE Communications Surveys Tutorials*, 19(3):1867–1901, 2017.

[13] S. R. Chowdhury, H. Bian, T. Bai, and R. Boutaba. A disaggregated packet processing architecture for network function virtualization. *IEEE Journal on Selected Areas in Communications*, 38(6):1075–1088, 2020.

[14] R. Cohen, L. Lewin-Eytan, J. S. Naor, and D. Raz. Near optimal placement of virtual network functions. In *INFOCOM 2015*.

[15] M. Ehrgott. *Multicriteria Optimization*. Springer, 2005.

[16] V. Eramo, E. Miucci, M. Ammar, and F. G. Lavacca. An approach for service function chain routing and virtual function network instance migration in network function virtualization architectures. *IEEE/ACM Transactions on Networking*, 25(4):2008–2025, 2017.

[17] X. Fei, F. Liu, Q. Zhang, H. Jin, and H. Hu. Paving the way for nfv acceleration: A taxonomy, survey and future directions. *ACM Comput. Surv.*, 53(4):1–42, 2020.

[18] P. X. Gao, A. Narayan, S. Karandikar, J. Carreira, S. Han, R. Agarwal, S. Ratnasamy, and S. Shenker. Network requirements for resource disaggregation. In *OSDI 2016*.

[19] A. Gushchin, A. Walid, and A. Tang. Scalable routing in sdn-enabled networks with consolidated middleboxes. In *ACM Hotmiddlebox*, 2015.

[20] H. Hantouti, N. Benamar, T. Taleb, and A. Laghrissi. Traffic steering for service function chaining. *IEEE Communications Surveys Tutorials*, 21(1):487–507, 2019.

[21] M. Huang, W. Liang, Z. Xu, and S. Guo. Efficient algorithms for throughput maximization in software-defined networks with consolidated middleboxes. *IEEE Transactions on Network and Service Management*, 14(3):631–645, 2017.

[22] N. Huin, B. Jaumard, and F. Giroire. Optimal network service chain provisioning. *IEEE/ACM Trans. on Netw.*, 26(3):1320–1333, June 2018.

[23] D. A. Joseph, A. Tavakoli, and I. Stoica. A policy-aware switching layer for data centers. In *ACM SIGCOMM 2008*.

[24] S. G. Kulkarni, W. Zhang, J. Hwang, S. Rajagopalan, K. K. Ramakrishnan, T. Wood, M. Arumaithurai, and X. Fu. Nfvnice: Dynamic backpressure and scheduling for nfv service chains. In *SIGCOMM 2017*.

[25] S. G. Kulkarni, W. Zhang, J. Hwang, S. Rajagopalan, K. K. Ramakrishnan, T. Wood, M. Arumaithurai, and X. Fu. Nfvnice: Dynamic backpressure and scheduling for nfv service chains. *IEEE/ACM Transactions on Networking*, 28(2):639–652, 2020.

[26] S. Li. Approximating capacitated k-median with $(1+\epsilon)k$ open facilities. In *the SODA 2016*.

[27] S. Li. On uniform capacitated k-median beyond the natural lp relaxation. *ACM Trans. Algorithms*, 13(2), January 2017.

[28] J.-H. Lin and J. S. Vitter. Approximation algorithms for geometric median problems. *Information Processing Letters*, 44(5):245–249, 1992.

[29] G. Liu, S. Guo, B. Li, and C. Chen. Joint traffic-aware consolidated middleboxes selection and routing in distributed sdns. *IEEE Transactions on Network and Service Management*, 18(2):1415–1429, 2021.

[30] G. Liu, S. Guo, P. Li, and L. Liu. Conmidbox: Consolidated middleboxes selection and routing in sdn/nfv-enabled networks. In *IEEE IPDPS 2020*.

[31] J. Liu, Y. Li, Y. Zhang, L. Su, and D. Jin. Improve service chaining performance with optimized middlebox placement. *IEEE Transactions on Services Computing*, 10(4):560–573, 2017.

[32] Y. Mao, X. Shang, and Y. Yang. Joint resource management and flow scheduling for sfc deployment in hybrid edge-and-cloud network. In *IEEE INFOCOM 2022*.

[33] R. J. Martins, C. B. Both, J. A. Wickboldt, and L. Z. Granville. Virtual network functions migration cost: from identification to prediction. *Computer Networks*, 181(9), 2020.

[34] N. Megiddo. Linear programming in linear time when the dimension is fixed. *Journal of the ACM*, 31(1):114–127, 1984.

[35] X. Meng, V. Pappas, and L. Zhang. Improving the scalability of data center networks with traffic-aware virtual machine placement. In *Proc. of IEEE INFOCOM 2010*.

[36] D. Merkel. Docker: Lightweight linux containers for consistent development and deployment. *Linux J.*, 2014(239), 2014.

[37] R. Mijumbi, J. Serrat, J. L. Gorricho, N. Bouten, F. D. Turck, and R. Boutaba. Network function virtualization: State-of-the-art and research challenges. *IEEE Communications Sur. and Tut.*, 18(1), 2015.

[38] B. Robert, L. Y. Chen, and E. Smirni. Data centers in the wild: A large performance study. IBM, Zurich, Switzerland, Rep., Z1204–002, 2012.

[39] A. Roy, H. Zeng, J. Bagga, G. Porter, and A. C. Snoeren. Inside the social network's (datacenter) network. In *SIGCOMM 2015*.

[40] V. Sekar, N. Egi, S. Ratnasamy, M. K. Reiter, and G. Shi. Design and implementation of a consolidated middlebox architecture. In *NSDI 2012*.

[41] R. Singh and S. Singh Gill. Edge ai: A survey. *Internet of Things and Cyber-Physical Systems*, 3:71–92, 2023.

[42] C. Sun, J. Bi, Z. Zheng, H. Yu, and H. Hu. Nfp: Enabling network function parallelism in nfv. In *SIGCOMM 2017*.

[43] X. Sun, N. Ansari, and R. Wang. Optimizing resource utilization of a data center. *IEEE Communications Surveys Tutorials*, 18(4):2822–2846, 2016.

[44] V. Tran, J. Sun, B. Tang, and D. Pan. Traffic-optimal virtual network function placement and migration in dynamic cloud data centers. In *IEEE IPDPS 2022*.

[45] W. Zhang, G. Liu, W. Zhang, N. Shah, P. Lopreiato, G. Todeschi, K. K. Ramakrishnan, and T. Wood. Opennetvm: A platform for high performance network service chains. In *ACM HotMiddlebox 2016*.

[46] Y. Zhang, N. Beheshti, L. Beliveau, G. Lefebvre, R. Manghirmalani, R. Mishra, R. Patney, M. Shirazipour, R. Subrahmaniam, C. Truchan, and M. Tatipamula. Steering: A software-defined networking for inline service chaining. In *IEEE ICNP 2013*.