Deadline-Aware SFC Orchestration under Demand Uncertainty

Minh Nguyen, Mahdi Dolati and Majid Ghaderi

***Abstract*—In network function virtualization, a service function chain (SFC) specifies a sequence of virtual network functions that user traffic has to traverse to realize a network service. The problem of SFC orchestration has been extensively studied in the literature. However, most existing works assume deterministic demands and resort to costly runtime resource reprovisioning to deal with dynamic demands. In this work, we formulate the deadline-aware co-located and geo-distributed SFC orchestration with demand uncertainty as robust optimization problems and develop exact and approximate algorithms to solve them. A key feature of our formulation is the consideration of end-to-end delay in service chains by carefully modeling load-independent propagation delay as well as load-dependent queueing and processing delays. To avoid frequent resource reprovisioning, our algorithms utilize uncertain demand knowledge to compute proactive SFC orchestrations that can withstand fluctuations in dynamic service demands. Extensive simulations are conducted to evaluate the performance of our algorithms in terms of ability to cope with demand fluctuations, scalability, and relative performance against other recent algorithms.**

***Index Terms*—Service function chain, Service orchestration, Demand uncertainty, End-to-end delay.**

I. Introduction

Background. Network function virtualization (NFV) refers to softwarization of network functions traditionally performed by specialized hardware appliances [1]. Typically, these software-based network functions, commonly referred to as virtual network functions (VNFs), are installed and run on virtual machines (VMs) hosted in a cloud-based NFV infrastructure (NFVI). To realize the benefits of NFV, several technical challenges need to be addressed. One important challenge is the orchestration of service function chains (SFCs) [2]. Each SFC specifies an ordered sequence of VNFs that user traffic has to traverse to realize a particular network service. *SFC orchestration* refers to the joint problem of deciding how many VNF instances to use, where in the NFVI to deploy these instances, and how to route traffic between them to meet certain objectives. A desirable objective is to minimize the cost of orchestrating the SFCs in the NFVI by provisioning just enough resources to satisfy service demands according to pre-specified service level agreements (SLAs).

Optimal provisioning of NFVI resources, nevertheless, is not trivial. The amount of resources a network service requires is determined based on the type of VNFs involved and the level of demand (*e.g.*, rate of traffic) for that service, which is only known at runtime. In particular, a key challenge in SFC orchestration is that service demands are dynamic and fluctuate over time. Thus, an orchestration algorithm has to either dynamically *react* to changes in demand by reprovisioning resources, or *proactively* provision the right amount of resources in advance to avoid the need for reprovisioning resources on the spot. While reactive algorithms are generally simpler to design, they result in unpredictable and often significant delays (see, *e.g.*, [3]) as they need to reprovision resources on-demand, e.g., migrate virtual machines or activate new servers. Most VNF operations are *strictly delay-sensitive* and run under stringent delay requirements. Any disruption to VNF operations results in poor performance and quality of service and has to be avoided. This paper focuses on developing SFC orchestration algorithms that utilize uncertain demand information to compute proactive SFC orchestration schemes that can cope with demand fluctuations, thereby avoiding frequent resource reprovisioning.

This work was supported by Alberta Innovates and Natural Sciences and Engineering Research Council of Canada.

Our Work. We consider the SFC orchestration problem with the objective of minimizing the deployment cost of VNFs. Specifically, given a list of SFCs representing the requested services and the NFVI resource information, our objective is to find the optimal deployment of VNFs required for these services that minimizes the total cost while respecting the service deadlines. Depending on the service type, VNFs deployed for a service can be either *co-located* within a single datacenter or *geo-distributed* over a dispersed cloud infrastructure [4], [5]. We formulate the co-located and geo-distributed service orchestration as constrained optimization problems, with each formulation being customized to capture the key considerations unique to each deployment scheme. Key considerations captured by our co-located SFC orchestration formulation include the dependence of the service delay on the traffic load and the sharing of a VNF instance between multiple service flows. For geo-distributed SFC orchestration, our formulation considers the dominance of propagation delays for communications between VNFs and the multi-domain nature of geo-distributed cloud infrastructures.

We develop exact and approximate algorithms to solve the SFC orchestration problem considering the two deployment schemes. To account for the demand fluctuations, our algorithms assume partial knowledge about future traffic demands, such as the nominal demands and the maximal demand deviations, which can often be obtained as part of SLAs or from past observations. This information is used by our algorithms to model fluctuations in the demand parameters as cardinality-constrained uncertainty sets [6]. By adjusting the cardinality of the uncertainty sets, the proposed algorithms can achieve any desired trade-off between deployment cost and proactivity

(to demand fluctuations) of the computed orchestration.

Our main contributions in this paper are summarized below:

- We define and formulate the co-located and geo-distributed SFC orchestration with the objective of minimizing deployment cost as integer optimization problems. To further account for the demand uncertainty, we use cardinality-constrained uncertainty sets to model the demand fluctuations and transform the original problem formulations to equivalent robust optimization programs.
- To deal with the non-linearity of the co-located SFC orchestration formulation, we apply various optimization techniques to transform the original formulation into a linear mixed-integer second-order cone program (MISOCP), which is tractable for small problem instances. For large problem instances, we develop an iterative algorithm that computes an approximate solution to the problem in polynomial time.
- For geo-distributed SFC orchestration, we design a multi-stage approximate algorithm that solves the placement and routing stages of the problem separately, while still accounting for the dependency between the two stages.
- We present extensive model-driven simulation results to study the behavior of our algorithms in small and large problem instances and demonstrate their ability to achieve any desired proactivity-cost trade-off. We further compare the performance of our algorithms against a baseline and two recent algorithms, namely FFCA [7] and MaxZ [8].

Paper Organization. The system models for co-located and geo-located problems are discussed in Section II. The problem formulations are presented in Sections III and IV, while our exact and approximation algorithms are described in Sections V and VI. Section VII is dedicated to performance evaluation of the algorithms. Finally, Section VIII reviews the relevant literature and Section IX concludes the paper.

II. System Model

This section details different components of our system models for co-located and geo-distributed SFC orchestration.

A. General Model

NFVI Model. The network infrastructure is modeled as a directed graph $\vec{G}(N, E)$, where N and E, respectively, are the sets of substrate nodes and links (see Fig. 3(b) for a datacenter-based NFVI). Each substrate node $n \in N$ has a certain number of processing units M_n available for hosting VNFs, with each unit provides a raw processing capacity of c_n. We assume that all physical links in the substrate network are full-duplex and model each as two equal-capacity directed links in opposite directions. Let r_e^{bwd} denote the aggregate rate of traffic on link $e \in E$, which should not be greater than the link capacity c_e^{bwd}. Further, let F denote the set of all VNFs that could potentially be deployed to the NFVI. Each VNF $f \in F$ has a processing coefficient σ_f, ranging between 0 and 1, that indicates its processing complexity (see [9] for experimental approaches for characterizing this value). Simple VNFs such as NAT, which do not require much processing, have their σ_f close to 1. On the other hand, processing-intensive VNFs such as DPI have smaller values of σ_f.

SFC Model. Let $\mathcal{U}$ denote the set of SFCs to be orchestrated in the NFVI. Each SFC $S \in \mathcal{U}$ is modeled as a virtual network represented by a directed graph $\vec{G} = (S_V, S_J)$, where S_V and S_J are, respectively, the sets of virtual nodes and links. Notice that this definition is the same as the one in RFC 7665 [10]. The virtual links' orientation describes the direction of the flow, while the virtual nodes represent the ingress/egress locations as well as the VNFs expected to process that flow (see Fig. 3(a) for examples of VNFs and SFCs). We use lower case letters $j \in S_J$ to denote an arbitrary virtual link and $v \in S_V$ to represent an arbitrary virtual node. The source and destination of each virtual link j are referred to as $j.source$ and $j.dest$, respectively. For each virtual node v, we define the set of all substrate nodes where v can be mapped to as $N_v \subseteq N$. The binary variable $a_{v,n}$ is used to indicate whether virtual node v is mapped to substrate node n. Also, let the variable $l_{j,e}$ denote if substrate link e is responsible for routing the traffic of virtual link j. Each virtual link j has a traffic demand r_j^{bwd}. We assume single-path routing, where each virtual link is mapped to a set of substrate links that together compose a path. For a list of typical virtual network function types and their corresponding applications, please refer to [9], [11].

B. Co-located SFC Orchestration

Deployment Scheme. In co-located SFC orchestration, the VNFs are all deployed within a single network infrastructure (*e.g.*, a datacenter). The substrate network in this case is a local area network with each substrate node being either a switch or a server. We have $N = N_{switch} \cup N_{server}$, in which N_{switch} and N_{server} denote the sets of switches and servers, respectively. We assume that only server nodes are capable of hosting VNFs (*i.e.*, $\forall S \in \mathcal{U}, \forall v \in S_V, N_v = N_{server}$).

Given full control of the network operator over the infrastructure in this context, we assume that a VNF scheduling mechanism can be adopted (*e.g.*, [12]) to allow a VNF instance to be shared between multiple service flows. The optimal number of VNF instances to deploy is determined as part of the optimization. Let the binary variable $i_{f,n}$ indicate if server n hosts an instance of VNF f. Here, each processing unit at a server node corresponds to one CPU core. Given the processing rate requirement $r_{f,n}^{proc}$ for VNF instance f on server n, we allocate $m_{f,n}$ cores to the VM that runs the function f. Unlike previous works which assume that the processing rate requirement is fixed and solely dependent on the type of VNF (*e.g.*, [7], [13–17]), *the processing rate requirement in our model is a function of the incoming traffic rate*. Multiple instances of VNF f can be deployed on different servers to speed up the processing, but at most, K_f servers can each host an instance of f at a time. The value of K_f is determined by the number of licenses that the network operator has acquired for VNF f.

Delay and Cost Models. The end-to-end delay for each SFC is computed as the sum of processing, queueing, and propagation delays over all of servers and links allocated to it. The processing delay of each VNF instance is approximated

by the response time of an $M/G/1$ processor sharing queue, as suggested by [18]. The processing delay $d_{f,n}^{proc}$ of VNF f on server n is determined by the allocated processing capacity $c_{f,n}^{proc}$ and its processing rate requirement $r_{f,n}^{proc}$. The processing capacity is dependent on the number of CPU cores allocated to the VM that runs the VNF instance, while the processing rate requirement is dependent on the incoming traffic rate. Each substrate link e is associated with a propagation delay d_e^{prp} and a queueing delay d_e^{bwd}. The propagation delay is always fixed. The queueing delay, on the other hand, is a function of the traffic load on the link. We estimate the queueing delay by the response time of an $M/M/1$ queue, as in [7], [8] and [19].

As a significant portion of the NFVI expense is due to the energy consumed for running the services (about 30-50% of the budget [20]), we consider the energy consumption as our cost metric to minimize. The energy consumption of both servers and switches is modeled by a linear function of their utilization [21]. While the switches are always active, we assume that the servers can be switched off if they do not host any VNF. The on/off status of server n is indicated by the binary variable o_n. We denote by P_n^{actv} the power consumption of active device n in idle state (*a.k.a.* idle power). When there is some load, the power consumption of device n, denoted by P_n, scales proportionally with its resource utilization. We let P_n^{max} denote the maximum power consumption of device n at full utilization. By considering future demand fluctuations, our model provisions sufficient amount of resources in one shot to avoid turning on and off physical machines at runtime.

C. Geo-distributed SFC Orchestration

Deployment Scheme. To deal with limited resources and geo-dispersed nature of users, some network services such as those in mobile cellular networks [4] are typically distributed over multiple datacenters, potentially managed by different infrastructure providers. The substrate network here is a wide area inter-datacenter network that spans a large geographical area. We assume each datacenter in this network offers M_n units of resources to host VNFs, with each unit providing a processing capacity of c_n. The total processing demand m_n (in terms of processing units) over all VNFs hosted by datacenter n should be at most M_n.

While in the co-located scheme, NFVI resources are controlled by a single operator, resources in the geo-distributed scheme are co-managed by different operators, *e.g.*, in a multi-tier architecture. For instance, an operator can provide *VNF-as-a-service* to several higher-tier operators without giving them any control over the VNF implementation or the underlying infrastructure [22]. Moreover, rather than hosting all VNFs locally, a service provider may outsource some to third-party VNF service providers who host their services in their datacenters. As such, not all VNFs are supported at each datacenter. We set N_v in a manner that restricts the mapping of each virtual node to only the datacenters where the corresponding VNF is supported. Further considering the fact that third-party VNF service providers generally do not support VNF sharing citing the security risks due to the multi-domain nature and management overhead cost [5], we do not allow a VNF instance to be shared between different services in this model.

Delay and Cost Models. As opposed to co-located NFVI, where the queueing and processing are primary sources of delay, the end-to-end delays in geo-distributed NFVI are dominated by the propagation delays [23]. We let d_e^{prp} denote the propagation delay on each link e. The end-to-end service delay is interpreted as the sum over propagation delays of all selected links to route the service flows, assuming single-path routing. Deploying a VNF instance at a datacenter will incur a cost proportional to the amount of resources being used. We assume there is an associated cost of P_n for each unit of resources being allocated at datacenter n. The primary goal of this deployment scheme, therefore, is to minimize the total cost of allocated resources across all datacenters.

III. Deterministic Problem Formulation

In this section, we formulate the co-located and geo-distributed SFC orchestration problems. Then, in the next section, we extend our formulations to cope with demand uncertainty.

A. Co-located SFC Orchestration

The objective is to compute the optimal service deployment scheme (including VNF placement – $a_{v,n}$, CPU allocation – $m_{f,n}$ and routing – $l_{j,e}$ decisions) that minimizes the energy consumption of the NFVI, given the service demand information from the SLAs, the network configurations along with the network resource information (*i.e.*, link and node capacities). The problem can be defined formally as a mixed-integer non-linear program (MINLP) as shown in Formulation 1.

Equation (A1) expresses the objective function of minimizing the total energy cost of the NFVI. More specifically, the energy consumption of servers and switches are expressed as a linear function of their capacity utilization as follows:

$$P_n = \begin{cases} o_n P_n^{actv} + \frac{\sum_{f \in F} m_{f,n}}{M_n}(P_n^{max} - P_n^{actv}), \forall n \in N_{server}, \\ P_n^{actv} + \frac{\sum_{e.dest=n} r_e^{bwd}}{c_n^{bwd}}(P_n^{max} - P_n^{actv}), \forall n \in N_{switch}. \end{cases}$$

Please note that the the generality of the model and decision variables allows us to consider other important objectives such as delay minimization. Constraint (A2) computes the load of each VNF instance as a function of placement variables $a_{v,n}$. Constraint (A3) computes the processing capacity $c_{f,n}^{proc}$ of VNF f on server n based on the number of allocated CPU cores $m_{f,n}$, the raw processing capacity of each core $c_{f,n}$, and the processing complexity of VNF f. Constraint (A4) ensures that the load of each VNF is not greater than its processing capacity. However, the total number of allocated cores at a server must be at most the number of available cores, as enforced by constraint (A5). Constraints (A6) and (A7) compute the processing delay associated with each deployed VNF instance and the queueing delay associated with each link. Constraint (A8) helps to set the variable w_e, which indicates if there is any traffic that is routed through link e. Specifically, constraint (A9) imposes a deadline θ_S on each service's end-to-end delay, which is calculated by summing up the queueing, processing and propagation delays. Constraint (A10) enforces that the number of deployed instances for a

Formulation 1 Co-located SFC Orchestration

$$Minimize \sum_{n} P_n \tag{A1}$$

$$r^{proc}_{f,n} = \sum_{S\in\mathcal{U}} \sum_{\substack{j\in S_J \\ t(j.dest)=f}} r^{bwd}_j \cdot a_{j.dest,n},\ \forall f \in F, \forall n \in N_{server}, \tag{A2}$$

$$c^{proc}_{f,n} = \sigma_f \cdot m_{f,n} \cdot c_n,\ \forall f \in F, \forall n \in N_{server}, \tag{A3}$$

$$r^{proc}_{f,n} \leq c^{proc}_{f,n},\ \forall f \in F, \forall n \in N_{server}, \tag{A4}$$

$$\sum_{f\in F} m_{f,n} \leq M_n,\ \forall n \in N_{server}, \tag{A5}$$

$$d^{bwd}_e = \frac{w_e}{c^{bwd}_e - r^{bwd}_e},\ \forall e \in E, \tag{A6}$$

$$d^{proc}_{f,n} = \frac{i_{f,n}}{c^{proc}_{f,n} - r^{proc}_{f,n}},\ \forall n \in N_{server}, \tag{A7}$$

$$w_e \geq l_{j,e},\ \forall S \in \mathcal{U}, \forall j \in S_J, \forall e \in E, \tag{A8}$$

$$\sum_{j\in S_J} \sum_{e\in E} l_{j,e}(d^{bwd}_e + d^{prp}_e) + \sum_{v\in S_v} \sum_{n\in N_{server}} \sum_{\substack{f\in F \\ t(v)=f}} a_{v,n} d^{proc}_{f,n} \leq \theta_S,\ \forall S \in \mathcal{U}, \tag{A9}$$

$$\sum_{n\in N_{server}} i_{f,n} \leq K_f,\ \forall f \in F, \tag{A10}$$

$$a_{v,n} \leq i_{f,n}, \forall S \in \mathcal{U}, \forall v \in S_V, \forall n \in N_{server}, \forall f \in F : t(v) = f, \tag{A11}$$

$$\sum_{n\in N_v} a_{v,n} = 1,\ \forall S \in \mathcal{U}, \forall v \in S_V, \tag{A12}$$

$$a_{v,n} = 0,\ \forall S \in \mathcal{U}, \forall v \in S_V, \forall n \in N : n \notin N_v, \tag{A13}$$

$$r^{bwd}_e = \sum_{S\in\mathcal{U}} \sum_{j\in S_J} r^{bwd}_j l_{j,e},\ \forall e \in E, \tag{A14}$$

$$r^{bwd}_e \leq c^{bwd}_e,\ \forall e \in E, \tag{A15}$$

$$\sum_{e:e.dest=n} l_{j,e} - \sum_{e:e.source=n} l_{j,e} = a_{j.dest,n} - a_{j.source,n}, \quad \forall n \in N, \forall S \in \mathcal{U}, \forall j \in S_J, \tag{A16}$$

$$o_n \geq i_{f,n},\quad \forall n \in N_{server}, \forall f \in F, \tag{A17}$$

$$P_n, c^{proc}_{f,n}, r^{proc}_{f,n}, r^{bwd}_e, d^{bwd}_e, d^{proc}_{f,n} \in \mathbb{R}^+,$$
$$l_{j,e}, a_{v,n}, o_n, i_{f,n}, w_e \in \{0,1\}, m_{f,n} \in \mathbb{N},$$
$$\forall S \in \mathcal{U}, \forall S \in \mathcal{U}, \forall j \in S_J, \forall v \in S_V, \forall e \in E., \forall f \in \mathcal{F}.$$

VNF should be no more than the number of licenses available for it. Except for the ingress and egress, each virtual node in an SFC denotes a VNF required for the service; constraint (A11) imposes that a virtual node must be mapped to a substrate node where the corresponding VNF is deployed (as indicated by the function $t(v)$). A further restriction is that a virtual node can only be mapped to a substrate node in its list of possible locations, as enforced by constraints (A12) and (A13). Constraint (A14) computes the flow rate of each link which is forced to respect its capacity in constraint (A15). Constraint (A16) is the standard equation for *multi-commodity single-path routing*. Finally, constraint (A17) identifies the active servers, those who host at least one VNF instance.

B. Geo-distributed SFC Orchestration

Given the inter-datacenter network topology with detailed information on the available resources and their costs at the datacenters along with the service demand information, our goal is to find the optimal placement and routing scheme (*i.e.*, finding $a_{v,n}$ and $l_{j,e}$) for the VNFs required to provide these services such that the total cost is minimized while guaranteeing the service deadlines. Formulation 2 presents a non-linear integer program (NLIP) to solve the problem.

Formulation 2 Geo-distributed SFC Orchestration

$$Minimize \sum_{n} P_n m_n \tag{B1}$$

$$\sum_{n\in N_v} a_{v,n} = 1,\quad \forall S \in \mathcal{U},\ \forall v \in S_v, \tag{B2}$$

$$m_n = \sum_{S\in\mathcal{U}} \sum_{v\in S_V, v\neq egress} \left\lceil \frac{r^{bwd}_{j:j.dest=v} a_{v,n}}{c_n \sigma_{t(v)}} \right\rceil,\ \forall n \in N, \tag{B3}$$

$$m_n \leq M_n,\ \forall n \in N, \tag{B4}$$

$$r^{bwd}_e = \sum_{S\in\mathcal{U}} \sum_{j\in S_J} r^{bwd}_j l_{j,e},\ \forall e \in E, \tag{B5}$$

$$r^{bwd}_e \leq c^{bwd}_e, \forall e \in E, \tag{B6}$$

$$\sum_{e:e.dest=n} l_{j,e} - \sum_{e:e.source=n} l_{j,e} = a_{j.dest,n} - a_{j.source,n}, \quad \forall n \in N, \forall S \in \mathcal{U}, \forall j \in S_J, \tag{B7}$$

$$\sum_{j\in S_J} \sum_{e\in E} l_{j,e} d^{prp}_e \leq \theta_S,\ \forall S \in \mathcal{U}, \tag{B8}$$

$$l_{j,e}, a_{v,n} \in \{0,1\}, m_n \in \mathbb{N}, r^{bwd}_e \in \mathbb{R}^+, \forall j \in S_J, \forall v \in S_V, \forall e \in E, \forall n \in N.$$

Equation (B1) states the objective function of minimizing the total cost for allocated resources over all datacenters in the network. Constraint (B2) enforces that every virtual node must be mapped to a substrate node (*i.e.*, every VNF request must be served by a datacenter where it is supported). The total processing demand, in terms of number of processing units, at each datacenter is computed by constraint (B3), while constraint (B4) ensures that this demand does not exceed the available number of resource units. Note that a flow destined to an egress node (*i.e.*, $j.dest = egress$) does not demand any processing and is thus omitted from demand calculation in (B3). For each link, constraint (B5) computes the total bandwidth demand over all service flows routed through it and constraint (B6) imposes the capacity limitation. Constraint (B7) expresses the regular flow conservation for single-path routing. A service deadline θ_S is imposed on the end-to-end propagation delay of each SFC S by constraint (B8).

IV. Coping with Demand Uncertainty

In this section, we consider modifying our deterministic formulations to incorporate demand uncertainty.

Demand Uncertainty Model. Each future traffic demand r^{bwd}_j is assumed to lie within an interval $[\bar{r}^{bwd}_j - \hat{r}^{bwd}_j, \bar{r}^{bwd}_j + \hat{r}^{bwd}_j]$, where $\bar{r}^{bwd}_j$ and $\hat{r}^{bwd}_j$ denote the nominal value of r^{bwd}_j and its maximum deviation, respectively. We assume that in the worst-case at most Γ demands can simultaneously deviate maximally from their nominal values, *i.e.*,

$$\sum_{j} \left| \frac{r^{bwd}_j - \bar{r}^{bwd}_j}{\hat{r}^{bwd}_j} \right| \leq \Gamma. \tag{R1}$$

Γ is a parameter that controls how proactive the model is to demand fluctuations. In practice, the accuracy of demand estimates affects the efficiency of the solutions produced by our algorithms. A small Γ results in a less proactive solution that may lead to frequent resource reprovisioning at runtime. By setting a large value for Γ, relative to the number of traffic demands, runtime reprovisioning can be completely avoided at the expense of a more costly solution. In this sense, Γ can be thought of as an *uncertainty budget* that expresses how

much additional resources the network operator is willing to allocate to account for user demand fluctuations.

Theorem 1. *To ensure deviation of any Γ demands does not make the model infeasible, it is necessary and sufficient that every resource constraint is protected against maximal deviations of any Γ demands associated with that constraint.*

Proof. It is trivial that when each constraint is individually protected from the worst-case deviation of Γ demands associated with it, then regardless of which Γ demands actually deviate, none of the resource constraints can be violated. On the other hand, it is necessary to protect every constraint from the worst-case deviation of Γ demands to ensure the feasibility of the model. Assume by contradiction that we do not need to consider the worst-case deviation of Γ demands at each constraint. Then there exists a set of Γ demands, which provides the worst-case demand deviation at a resource constraint but is not considered. If the Γ demands in this set are the only ones that deviate and that they deviate maximally, then the corresponding constraint could be violated, which contradicts our assumption. □

Robust Optimization. Theorem 1 implies that to account for the deviations of any Γ demands, we must consider the maximal Γ demand deviations associated with each capacity constraint. The problem is then reduced to an instance of Γ–robust optimization [6]. Specifically, we replace the processing and bandwidth demand computations in the original formulations (*i.e.*, (A2), (A14), (B3) and (B5)) by their robust counterparts. The robust counterpart of (A2) is expressed as follows:

$$r_{f,n}^{proc} = \sum_{S\in\mathcal{U}} \sum_{\substack{j\in S_J \\ t(j.dest)=f}} \bar{r}_j^{bwd} a_{j.dest,n} + \max_{|Q_{f,n}^{proc}|\le\Gamma} \sum_{\substack{j\in Q_{f,n}^{proc} \\ t(j.dest)=f}} \hat{r}_j^{bwd} a_{j.dest,n}, \ \forall n\in N, \forall f\in F, \tag{R2}$$

where $Q_{f,n}^{proc} \subseteq \bigcup_{S\in\mathcal{U}} S_J$ denotes the uncertainty set of Γ virtual links selected by the model to take the maximum deviations. Following the procedure in [6], we take the duality of the inner maximization problem and reformulate (R2) as the following pair of linear constraints:

$$r_{f,n}^{proc} = \Gamma\alpha_{f,n} + \sum_{S\in U} \sum_{\substack{j\in S_J \\ t(j.dest)=f}} \bar{r}_j^{bwd} a_{j.dest,n} + \beta_{j,f,n}, \quad \forall f\in F, \forall n\in N_{server}, \tag{R3}$$

$$\alpha_{f,n} + \beta_{j,f,n} \ge \hat{r}_j^{bwd} a_{j.dest,n}, \quad \forall f\in F, \forall n\in N_{server}, \forall S\in\mathcal{U}, \forall j\in S_J : t(j.dest)=f. \tag{R4}$$

In the same manner, the linear robust counterparts for (A14) and (B5) can be derived to be:

$$r_e^{bwd} = \Gamma\gamma_e + \sum_{S\in\mathcal{U}} \sum_{j\in S_J} l_{j,e}\bar{r}_j^{bwd} + \zeta_{j,e}, \ \forall e\in E, \tag{R5}$$

$$\gamma_e + \zeta_{j,e} \ge \hat{r}_j^{bwd} l_{j,e}, \ \forall e\in E, \forall S\in\mathcal{U}, \forall j\in S_J. \tag{R6}$$

For (B3), both the uncertain coefficient and placement decision variable $a_{v,n}$ are inside a ceiling function, making the direct conversion to robust counterpart challenging. Instead, we move the placement variable $a_{v,n}$ out of the ceiling function and shift the uncertainty from $r_{j:j.dest=v}^{bwd}$ to $\lceil r_{j:j.dest=v}^{bwd}/c_n\sigma_{t(v)}\rceil = m_{v,n}$, the number of processing units required by virtual node v as if v is serviced at datacenter n. Constraint (B3) then can be rewritten as follows:

$$m_n = \sum_{S\in\mathcal{U}} \sum_{v\in S_v, v\neq egress} m_{v,n} a_{v,n}, \ \forall n\in N, \tag{R7}$$

with it's linear robust counterpart being:

$$m_n = \Gamma\psi_n + \sum_{S\in\mathcal{U}} \sum_{v\in S_v, v\neq egress} \bar{m}_{v,n} a_{v,n} + \delta_{v,n}, \ \forall n\in N, \tag{R8}$$

$$\psi_n + \delta_{v,n} \ge \hat{m}_{v,n} a_{v,n}, \ \forall n\in N, \forall S\in\mathcal{U}, \forall v\in S_V, \tag{R9}$$

in which $\lceil \bar{r}_{j:j.dest=v}^{bwd}/c_n\sigma_{t(v)}\rceil = \bar{m}_{v,n}$ and $\lceil \hat{r}_{j:j.dest=v}^{bwd}/c_n\sigma_{t(v)}\rceil = \hat{m}_{v,n}$. The optimization variables $\alpha_{f,n}$, $\beta_{j,f,n}$, ψ_n, $\delta_{v,n}$, γ_e, and $\zeta_{j,e}$ are dual variables introduced when deriving the robust counterparts.

When (R1) does not hold, Theorem 2 presents the upper bound on the likelihood that a capacity resource constraint is violated. The proof for Theorem 2 extends the original proof in [6] considering the specific structure of our problem to derive a tighter upper bound on the capacity violation probability.

Theorem 2. *Let Pr_i^{vio} denote the probability that the resource constraint i is violated. Assume that the traffic demands are symmetrically distributed over their corresponding ranges, we have:*

$$Pr_i^{vio} \le \begin{cases} 0 & \text{if } \Gamma \ge M_i, \\ \exp\left(-\dfrac{\Gamma^2}{2M_i}\right) & \text{if } 1\le\Gamma<M_i, \end{cases} \tag{R10}$$

where,

$$M_i = \left\lfloor \frac{\textit{capacity of resource } i}{\textit{minimum nominal demand}} \right\rfloor \le \sum_{S\in\mathcal{U}} |S_J|. \tag{R11}$$

Proof. In the first case, when $\Gamma \ge M_i$, as the model guarantees to provide accommodation for maximum deviations of any combination of demands that can be possibly mapped to resource i, the resource constraint will never be violated, thus $Pr_i^{vio} = 0$. For $1 \le \Gamma \le M_i$, assume $x_{j,i}^*$ is the decision made by the robust model on whether to map demand j to resource i, we know the following relation holds from [6]:

$$Pr_i^{vio} \le \exp\left(\sum_{j\in\{\bigcup_{S\in\mathcal{U}} S_J\}} \frac{\rho^2\kappa_{j,i}^2}{2} - \rho\Gamma_i \right), \tag{R12}$$

where,

$$\kappa_{j,i} = \begin{cases} 1 & \text{if } j\in Q_i^*, \\ \dfrac{\hat{r}_j x_{j,i}^*}{\hat{r}_{\tau^*} x_{\tau^*,i}^*} & \text{if } j\in C_i\backslash Q_i^*, \end{cases} \tag{R13}$$

$$\tau^* = \underset{\tau\in Q_i^*}{\operatorname{argmin}}\{\hat{r}_\tau x_{\tau,i}^*\}. \tag{R14}$$

We have ρ is a constant, C_i is the set of uncertain coefficients associated with resource constraint i while $Q_i^* \subseteq C_i$ is the subset of coefficients that are selected to take their maximum deviations, and τ^* indicates the demand with the smallest deviation magnitude that is selected by the optimization model to include in Q_i^*. In our formulations, $x_{j,i}^*$ corresponds to the decision variables $a_{v,n}$ and $l_{j,e}$, which denote the mapping of virtual nodes and links, respectively. Also, the set of uncertain coefficients associated with each resource constraint is the set

of all traffic demands, *i.e.*, $\forall_i\ C_i = \{\bigcup_{S\in\mathcal{U}} S_J\}$.

Assume $1 \leq \Gamma < M_i$, we have $|Q_i^*| \leq \min\{\Gamma, M_i\} = \Gamma$. Also, for every demand $j \in Q_i^*$, we know that its corresponding decision variable $x_{j,i}$ must be set to one, since otherwise it would not be included in Q_i^*. Then, because at most M_i demands can be mapped to resource i (*i.e.*, $\sum x_{j,i}^* \leq M_i$), for those demands $j \in C_i \backslash Q_i^*$ that are not selected to take their maximum deviations, no more than $M_i - \Gamma$ decision variables $x_{j,i}$ can be set to one. Therefore,

$$\sum_{j\in C_i\backslash Q_i^*} \frac{\hat{r}_j x_{j,i}^*}{\hat{r}_{\tau^*} x_{\tau^*,i}^*} \leq M_i - \Gamma \tag{R15}$$

as $\hat{r}_j/\hat{r}_{\tau^*} \leq 1$ by the definition of τ^* in (R14), we obtain that

$$\sum_{j\in C_i} \kappa_{j,i} \leq M_i. \tag{R16}$$

Combining (R16) and (R12), it is obtained that

$$Pr_i^{vio} \leq \exp\left(\frac{M_i\rho^2}{2} - \rho\Gamma_i\right) = \exp\left(-\frac{\Gamma_i^2}{2M_i}\right), \tag{R17}$$

in which the parameter constant ρ is set to Γ_i/M_i. □

V. Scalable Co-located SFC Orchestration

In this section, we focus on developing algorithms that can be used to solve large instances of the co-located problem.

A. Reduction to MISOCP

We note that the delay constraints (A6), (A7) and (A8) in Formulation 1 are non-linear. In the following, we show how these constraints can be reformulated in order to transform the model into a MISOCP, which can be solved using standard optimization solvers such as Gurobi [24].

Let $\tilde{d}_e^{bwd}$ and h_e^{bwd}, respectively, denote the upper bound on the queueing delay and the residual capacity of link e. Then (A6) can be rewritten as follows:

$$\tilde{d}_e^{bwd} h_e^{bwd} \geq w_e,\ \ \forall e \in E, \tag{A19}$$
$$h_e^{bwd} = c_e^{bwd} - r_e^{bwd},\ \ \forall e \in E. \tag{A20}$$

The product term in (A19) can be expressed as a sum of squares as follows:

$$2w_e^2 + (\tilde{d}_e^{bwd})^2 + (h_e^{bwd})^2 \leq (\tilde{d}_e^{bwd} + h_e^{bwd})^2,\ \ \forall e \in E. \tag{A21}$$

Notice that since w_e is a binary variable, it can be conveniently replaced by its square, which consequently transforms the constraint into a *second-order conic constraint* (SOCP). In the same way, we can rewrite (A7) as follows:

$$2i_{f,n}^2 + (\tilde{d}_{f,n}^{proc})^2 + (h_{f,n}^{proc})^2 \leq (\tilde{d}_{f,n}^{proc} + h_{f,n}^{proc})^2, \quad \forall f \in F, \forall n \in N_{server}, \tag{A22}$$
$$h_{f,n}^{proc} = c_{f,n}^{proc} - r_{f,n}^{proc},\ \ \forall f \in F, \forall n \in N_{server}, \tag{A23}$$

where $\tilde{d}_{f,n}^{proc}$ and $h_{f,n}^{proc}$ are, respectively, the upper bound on the processing delay and the unused portion of the allocated processing capacity of VNF f on server n.

Next, we substitute the individual delay components in (A8) by their upper bounds:

$$\sum_{j\in S_J}\sum_{e\in E} l_{j,e}(\tilde{d}_e^{bwd} + d_e^{prp}) + \sum_{v\in S_v}\sum_{n\in N_{server}}\sum_{\substack{f\in F\\ t(v)=f}} a_{v,n}\tilde{d}_{f,n}^{proc} \leq \theta_S,\ \ \forall S \in \mathcal{U}. \tag{A24}$$

We use the *big-M* approach to linearize the products of binary and continuous variables in (A24). We define $\tilde{d}_{j,e}^{bwd} = l_{j,e}\tilde{d}_e^{bwd}$ as the upper bound on the queueing delay experienced by traffic flow j at link e. Assume that $\tilde{d}_e^{bwd}$ is upper bounded by a constant M_1, we then can express $\tilde{d}_{j,e}^{bwd} = l_{j,e}\tilde{d}_e^{bwd}$ by the following set of linear constraints:

$$\tilde{d}_e^{bwd} \leq \tilde{d}_{j,e}^{bwd} + (1 - l_{j,e})M_1,\ \ \forall e \in E, \forall S \in \mathcal{U}, \forall j \in S_J, \tag{A25}$$
$$\tilde{d}_{j,e}^{bwd} \leq l_{j,e}M_1,\ \ \forall e \in E, \forall S \in \mathcal{U}, \forall j \in S_J, \tag{A26}$$
$$\tilde{d}_{j,e}^{bwd} \leq \tilde{d}_e^{bwd},\ \ \forall e \in E, \forall S \in \mathcal{U}, \forall j \in S_J. \tag{A27}$$

Similarly, given a VNF request v of type f (*i.e.*, $t(v) = f$), we define $\tilde{d}_{v,n}^{proc} = a_{v,n}\tilde{d}_{f,n}^{proc}$ as the upper bound on its processing delay at server n. Under the assumption that $\tilde{d}_{f,n}^{proc}$ is upper bounded by a constant M_2, we can linearly express $\tilde{d}_{v,n}^{proc} = a_{v,n}\tilde{d}_{f,n}^{proc}$ as follows:

$$\tilde{d}_{f,n}^{proc} \leq \tilde{d}_{v,n}^{proc} + (1 - a_{v,n})M_2, \quad \forall n \in N_{server}, \forall f \in F, \forall S \in \mathcal{U}, \forall v \in S_V : t(v) = f, \tag{A28}$$
$$\tilde{d}_{v,n}^{proc} \leq a_{v,n}M_2,\ \ \forall n \in N_{server}, \forall f \in F, \forall S \in \mathcal{U}, \forall v \in S_V, \tag{A29}$$
$$\tilde{d}_{v,n}^{proc} \leq \tilde{d}_{f,n}^{proc},\ \ \forall n \in N_{server}, \forall f \in F, \forall S \in \mathcal{U}, \forall v \in S_V : t(v) = f. \tag{A30}$$

For this to work, M_1 and M_2 should be set to sufficiently large values. Although we rely on the program to compute $\tilde{d}_{v,n}^{proc}$ and $\tilde{d}_{j,e}^{bwd}$, we know that these individual delay components should not be greater than the deadline θ_S of SFC S to which v and j belong. Thus, by setting $M_1 = M_2 = \max_S \theta_S$, we ensure that M_1 and M_2 are always greater than any $\tilde{d}_{v,n}^{proc}$ and $\tilde{d}_{j,e}^{bwd}$ in a feasible solution. The end-to-end delay requirement now can be expressed as the following linear constraint:

$$\sum_{j\in S_J}\sum_{e\in E} \tilde{d}_{j,e}^{bwd} + l_{j,e}d_e^{prp} + \sum_{v\in S_v}\sum_{n\in N_{server}}\sum_{\substack{f\in F\\ t(v)=f}} \tilde{d}_{v,n}^{proc} \leq \theta_S,\ \ \forall S \in \mathcal{U}. \tag{A31}$$

The modifications we make deal solely with the delay computations and do not have any effect on the decisions of where to deploy the VNFs or which paths to use for routing the traffic. As such, the optimal solution remains intact.

To further simplify the model, we apply the *penalty method* to convert constraint (A31) into a term in the objective, while still keeping the integrality of the model. Define the auxiliary variable $z_s \geq 0$ as follows:

$$\sum_{j\in S_J}\sum_{e\in E} \tilde{d}_{j,e}^{bwd} + l_{j,e}d_e^{prp} + \sum_{v\in S_v}\sum_{n\in N_{server}}\sum_{\substack{f\in F\\ t(v)=f}} \tilde{d}_{v,n}^{proc} - \theta_S \leq z_S, \forall S \in \mathcal{U}. \tag{A32}$$

A new term that captures the delay requirement is then added to the objective:

$$\text{Minimize} \sum_{n\in N} P_n + \lambda \sum_{S\in\mathcal{U}} z_S, \tag{A33}$$

where λ is a parameter that models the importance of fulfilling the hard deadlines. Specifically, by setting λ to a

sufficiently large value, the model *is forced to meet the hard deadlines* (*i.e.*, $z_S \to 0$ as $\lambda \to \infty$). In this case, the total energy consumption $\sum_n P_n$ returned by the modified model will be no different from the original. Otherwise, by setting λ to smaller values, the model has the flexibility to trade-off some delay performance for reduced provisioning cost.

Our algorithm called *Service Orchestration with DeadLine* (SODL) to solve the co-located SFC orchestration problem (exactly) as a MISOCP is given in Algorithm 1.

Algorithm 1 SODL (Exact Co-located SFC Orchestration)

Minimize $\sum_{n \in N} P_n + \lambda \sum_{S \in \mathcal{U}} z_S$

s.t. resource cstrs: (A3)–(A5), (R3) and (R4) energy cstrs: (A17)
routing cstrs: (A15), (A16), (R5) and (R6) placement cstrs: (A10)–(A13)
delay cstrs: (A19)–(A32).

B. Approximate Service Orchestration

The presence of integer variables in SODL makes it computationally intractable for large problem instances. In this section, we develop an orchestration algorithm called xSODL (approXimate SODL) that employs iterative rounding [8] to compute an approximate solution to the problem. To apply iterative rounding, we first relax all integer variables in the MISOCP (*i.e.*, $a_{v,n}$, $l_{j,e}$, $i_{f,n}$, w_e, o_n and $m_{f,n}$) transforming it into a SOCP, which is a special class of quadratic programming that can be solved in polynomial time [25]. The relaxed model is then solved iteratively to obtain fractional solutions. In each iteration, we selectively round a subset of variables *in a manner that respects all constraints in the model* and fix their values in subsequent iterations. The procedure terminates when all relaxed integer variables are fixed. The details of these steps are given in Algorithm 2 and further explained below. Table I summarizes the notations used in the algorithm.

TABLE I: Summary of Notations Used in xSODL

Notation	Description
$\Upsilon \in \mathbb{N}$	Number of virtual nodes (VNF requests) that are mapped per iteration
$\mathcal{K}_f \in \mathbb{N}$	Number of unused licenses for VNF f
$\mathcal{M}$	The relaxed model
$\mathcal{M}_{sol}$	Solutions returned by solving $\mathcal{M}$
$\mathcal{V}$	Set of virtual nodes (VNF requests) still need to be mapped
$\mathcal{P}_{n,f}$	Set of VNFs requests that are mapped to VNF instance f at server n
$\mathcal{F}_n$	Set of *different* VNFs (types) at server n
$\mathcal{J}$	Set of virtual links (traffic flows between VNFs) still need to be routed
$\mathcal{L}_e$	Set of flows that are routed through substrate link e

Algorithm 2 xSODL

Input: $\mathcal{M}$, Γ, Υ **Return:** M_{sol}
1: $\mathcal{V} \leftarrow \{v \mid \forall S \in \mathcal{U}, \forall v \in S_V, v \neq ingress \wedge v \neq egress\}$
2: $\mathcal{P}_{n,f} \leftarrow \{\}\ \forall n \in N_{server}, \forall f \in F$
3: $\mathcal{F}_n \leftarrow \{\}\ \forall n \in N_{server}$
4: $\mathcal{J} \leftarrow \{j \mid \forall S \in \mathcal{U}, \forall j \in S_J\}$
5: $\mathcal{L}_e \leftarrow \{\}\ \forall e \in E$
6: $\mathcal{K}_f \leftarrow K_f\ \forall f \in F$
7: `RestrictPlacement`$(\mathcal{M}, \mathcal{P}, \mathcal{V}, \Gamma, \mathcal{F}, \mathcal{K})$
8: `RestrictRouting`$(\mathcal{M}, \mathcal{J}, \mathcal{L}, \Gamma)$
9: $\mathcal{M}_{sol} \leftarrow$ Solves the relaxed program ▷ Obtain an initial solution
10: **while** $|\mathcal{V}| > 0$ **do**
11: **for** $i \leftarrow 1$ to $\min\{\Upsilon, |\mathcal{V}|\}$ **do**
12: $(v, n) \leftarrow \text{argmax}_{(v,n)}\{a_{v,n} | \forall a_{v,n} \in \mathcal{M}_{sol}, v \notin \mathcal{V} \wedge n \in N_v\}$
13: $f \leftarrow v.type$
14: $\mathcal{M}.addConstr(a_{v,n} = 1)$ ▷ Fix the placement
15: $\mathcal{P}_{n,f}.orderInsert(v)$ ▷ Record where v is mapped
16: **if** $f \notin \mathcal{F}_n$ **then**
17: $\mathcal{F}_n.append(f)$ ▷ Keep track of the deployed VNF instances
18: $\mathcal{K}_f \leftarrow \mathcal{K}_f - 1$ ▷ Update the number of remaining licenses
19: **end if**
20: `RestrictPlacement`$(\mathcal{M}, \mathcal{P}, \mathcal{V}, \Gamma, \mathcal{F}, \mathcal{K})$
21: $\mathcal{M}.updateConstr((A5), \sum_{f \in F} m_{f,n} \leq M_n - |\mathcal{F}_n| + 1)$ ▷ CPU reservation
22: $\mathcal{V}.remove(v)$
23: **end for**
24: $\mathcal{M}_{sol} \leftarrow$ Solves the relaxed program
25: **for** $j \in S_J, \forall S \in \mathcal{U}$ **do**
26: **if** $a_{j.source,n_1} = 1 \wedge a_{j.dest,n_2} = 1 \wedge j \in \mathcal{J}$ **then**
27: `Route`$(\mathcal{M}, n_1, n_2, j, \mathcal{L})$ ▷ Route flow j from server n_1 to server n_2
28: $\mathcal{J}.remove(j)$
29: **end if**
30: **end for**
31: `RestrictRouting`$(\mathcal{M}, \mathcal{J}, \mathcal{L}, \Gamma)$
32: **end while**
33: $\mathcal{M}_{sol} \leftarrow$ Solves the relaxed program
34: **for** $m_{f,n} \in \mathcal{M}_{sol}$ **do**
35: $\mathcal{M}.addConstr(m_{f,n} = \lceil m_{f,n} \rceil)$ ▷ Fix the CPU allocation
36: **end for**
37: $\mathcal{M}.updateConstr((A4), \sum_{f \in F} m_{f,n} \leq M_n)$ ▷ Restore servers' capacities
38: $\mathcal{M}_{sol} \leftarrow$ Solves the relaxed program
39: **return** $\mathcal{M}_{sol}$

Fractional Placement. The mapping of virtual nodes to the substrate nodes is given by the variable $a_{v,n}$. We define the list $\mathcal{V}$ which keeps track of virtual nodes that have not been mapped. In each iteration of the main loop (lines 10-32), considering only the mappings (v, n) where $v \in \mathcal{V}$ and $n \in N_v$, the algorithm selects the top Υ variables among $a_{v,n}$ variables that have the largest fractional values (line 12) and fixes them to one by adding an equality constraint to the model (line 14). For each virtual node being mapped, xSODL keeps track of where it is mapped (line 15), the list of different VNFs that have been instantiated on the selected server (line 17) and the remaining number of licenses (line 18). This information is later used by `RestrictPlacement` to update N_v and the model (line 21). The virtual nodes that have been mapped are also removed from $\mathcal{V}$ (line 22), preventing them from being re-picked in future iterations. Once all virtual nodes have been mapped (*i.e.*, $\mathcal{V} = \emptyset$), the main loop of the algorithm (lines 10 - 32) terminates. The parameter Υ controls the number of virtual nodes to be mapped in each iteration. The more virtual nodes that are mapped in each iteration, the faster the algorithm will run. Therefore, by increasing Υ, we can make the algorithm scalable to problems involving a large number of VNF requests. However, as it is later shown in Section VII, the reduction in runtime achieved by increasing Υ comes at the expense of higher energy provisioning cost.

Placement Feasibility. Rounding a fractional assignment to an integral one can make the model infeasible. To avoid such scenarios, after each successful assignment, `RestrictPlacement` iterates through every possible mapping (v, n) of each virtual node v that has not yet been considered (*i.e.*, $v \notin \mathcal{V}$) and eliminates mappings that, if chosen, will make the model infeasible (*i.e.*, by updating N_v). There are two cases to consider, as follows:

Case (I). The server does not have enough capacity to support the full traffic load of the fractionally assigned virtual nodes. To avoid this case, the algorithm keeps track of all virtual nodes that have been mapped to each server so far. These virtual nodes are sorted in descending order of their maximum demand deviations. The top Γ virtual nodes are assigned their worst-case demands. When checking if it is feasible to assign virtual node v to server n, assuming that v is already placed at n, `RestrictPlacement` sums up the CPU demands

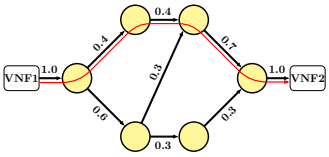


Fig. 1: The path that can carry the largest fractional amount of flow is the one that is most likely to cost the least.

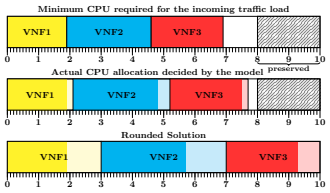


Fig. 2: Reserving 2 CPUs on a server with 3 VNF instances ensures that the rounding does not violate the capacity.

of all VNFs and verifies that it is less than $M_n - |\mathcal{F}_n| + 1$, where M_n is the total number of available CPU cores and $|\mathcal{F}_n|$ is the number of different VNFs on server n.

Case (II). The virtual node is fractionally assigned to a server where no VNF of the same type has been instantiated while all licenses have been used. To avoid this case, the algorithm records the number of licenses that have been issued for each type of VNF. When all licenses of VNF f have been used, virtual nodes of type f can only be mapped to servers where an instance of f already exists.

Fractional Routing. The mapping of virtual links (flows) to substrate links is indicated by the variable $l_{j,e}$. As the binary constraint on $l_{j,e}$ is relaxed, a flow can be split into multiple sub-flows and routed through different paths. When two ends of a virtual link have been mapped, as a result of solving the relaxed model on line 24, we obtain an optimal multi-path routing solution for the corresponding flow. Out of the paths returned by the relaxed model, `Route` chooses the path that carries the largest fractional amount of flow to route the traffic (line 27). The problem is referred to in the literature as the *Widest Path* problem, and can be solved using a variant of the Dijkstra's algorithm [26]. To select a path, all fractional flow assignments on the links that constituted that path are rounded to one. Fig. 1 shows an example of how a single-path solution is constructed from the fractional flow assignments.

Routing Feasibility. As with mapping the virtual nodes, the algorithm keeps track of all virtual links that have not been mapped yet in $\mathcal{J}$. In each iteration, only the virtual links that are still in $\mathcal{J}$ are considered. `RestrictRouting` iterates through every possible mapping of each virtual link in $\mathcal{J}$ and excludes mappings that make the model infeasible. This is carried out by directly adding constraints to the relaxed model to prohibit it from routing fractions of flows through links that do not have enough capacity to accommodate the full flows.

Fractional CPU Allocation. The last variable to fix in the algorithm is $m_{f,n}$, which determines the number of CPU cores allocated to each deployed VNF instance. Unlike $a_{v,n}$ and $j_{j,e}$, variable $m_{f,n}$ cannot be fixed on-the-go when a virtual node is mapped since more virtual nodes in subsequent iterations could be mapped to the same VNF instance increasing the CPU demand. Recall that the number of allocated CPU cores to a VNF is a function of the demand for that VNF. The only option, therefore, is to wait until all virtual nodes and links have been mapped to perform rounding on $m_{f,n}$. Specifically, after all variables $a_{v,n}$ and $j_{j,e}$ have been fixed, the algorithm reruns the relaxed model to obtain the optimal fractional CPU allocations (line 33), and then rounds up the computed fractional solutions (lines 34-36).

CPU Allocation Feasibility. To ensure that the rounding of allocated fractional CPU cores does not violate the server capacity constraint, we reserve a certain number of CPU cores on each active server for the rounding (line 21). The number of CPU cores to reserve is determined based on the number of VNF instances $|\mathcal{F}_n|$ on the server.

Theorem 3. *Preserving $|\mathcal{F}_n| - 1$ CPU cores ensures that the capacity of server n will not be violated due to the rounding of the allocated fractional CPU cores.*

Proof. Without loss of generality, assume $m_{f,n}$ is the fractional amount of CPU cores allocated to a VNF instance f at substrate node n. By model definition, it is enforced that $\sum_{f\in\mathcal{F}_n} m_{f,n} \leq M_n - (|\mathcal{F}_n| - 1)$. Since certainly $\lceil m_{f,n}\rceil \leq m_{f,n}+1$, it follows that $m_{f_x,n}+\sum_{f\in\mathcal{F}_n-\{f_x\}}\lceil m_{f,n}\rceil \leq M_n - (|\mathcal{F}_n| - 1) + (|\mathcal{F}_n| - 1)$, and thus $\sum_{f\in\mathcal{F}_n}\lceil m_{f,n}\rceil \leq M_n$. □

Once the algorithm has finished fixing $m_{f,n}$ (line 35), it restores the capacities of the servers and reruns the model (lines 37-38) to apply the fixed $m_{f,n}$ to all derived variables before returning the final solution. Fig. 2 shows an example demonstrating how our CPU reservation and rounding schemes work.

VI. Scalable Geo-distributed SFC Orchestration

In this section, we devise a multi-stage algorithm named xGSODL that, through separately solving the placement and routing in an iterative manner, computes an approximate orchestration solution to the problem. Although the placement and routing are solved separately in each iteration, xGSODL considers the scenario where a feasible placement with minimal provisioning cost may result in no feasible routing scheme and provides a mechanism to cope with such an inter-stage dependency. Algorithm 3 details the steps of xGSODL while Table II summarizes the notations used therein.

A. Placement

VNF Placement as Generalized Assignment. In the first stage of the algorithm (lines 5-13), we focus only on finding a VNF placement that minimizes the total cost. The model for VNF placement is comprised of the objective function (B1) together with constraints (B2)–(B4) in the original formulation. We refer to this VNF placement model in xGSODL as $\mathcal{M}_{place}$. The VNF placement is a form of the generalized assignment problem for which finding just a feasible solution is NP-complete. A generalized assignment instance for VNF placement can be modeled as a bipartite graph. In the bipartite graph, there exists a link $e(v,n)$ connecting a virtual node v to a substrate node n if n is in the list of possible locations of v, $n \in N_v$. An assignment $e(v,n)$ consumes $m_{v,n}$ units of capacity while incurring a cost $m_{v,n}P_n$. Our task is to find a minimal-cost assignment scheme that honors the capacity constraints. The problem has been extensively studied with a number of solution approaches reported in the literature. Considering only the nominal traffic demands, we employ

TABLE II: Summary of Notations Used in xGSODL

Notation	Description
$\mathcal{M}_{place}$	Relaxed VNF placement model
$\mathcal{M}_{route}$	Relaxed VNF routing model
N_{sorted}	Sorted DCs in descending order of their c_n/P_n
$\mathcal{SP}_{n_1,n_2}$	Set of paths between DC n_1 and n_2
$\mathcal{P}_j$	Set of paths for routing flow j
$\mathcal{P}_e$	Set of paths that use substrate link e
$\mathcal{J}_e$	Set of flows that are routed through substrate link e
$\mathcal{P}_n$	Set of virtual nodes (VNF requests) that are mapped to n
E^{irr}	Set of links whose capacities cannot be met in the irreducible system
S^{irr}	Set of SFCs whose deadlines cannot be satisfied in the irreducible system

Algorithm 3 xGSODL

Input: $\mathcal{M}_{place}, \Gamma, k$ **Return:** $(\{\mathcal{P}_n|\forall n \in N\}, \{\mathcal{J}_e|\forall e \in E\})$

1: $N_{sorted} \leftarrow \left\{n \;\middle|\; \forall n \in N,\ \frac{c_n}{P_n} \geq \frac{c_{n+1}}{P_{n+1}}\right\}$ ▷ Sort the DCs in descending order
2: $\mathcal{SP}_{n_1,n_2} \leftarrow$ `YenKSP`$(n_1, n_2),\ \forall (n_1, n_2) \in N$ ▷ Compute the k-shortest paths
3: $N_v \leftarrow \{n \mid m_{v,n} \leq M_n, \forall n \in N_v\}\ \forall v \in N_v$
4: **while** $True$ **do**
5: $\{\mathcal{P}_n|\forall n \in N\} \leftarrow$ `GAIterativeRounding`$(N_v, \mathcal{M}_{place})$ ▷ Find initial placement
6: **for** $n \in N_{sorted}$ **do** ▷ Check the DC capacity and perform migration if necessary
7: **if** $\sum_{v\in\mathcal{P}_n} \hat{m}_{v,n} + \max_{\mathcal{P}'_n\subseteq\mathcal{P}_n, |\mathcal{P}'_n|\leq\Gamma} \sum_{v\in\mathcal{P}'} \hat{m}_{v,n} > M_n$ **then**
8: $\mathcal{P}_n, \mathcal{P}_n^{mig} \leftarrow$ Solves (B9) with $c_n = c_n - \max_{\mathcal{P}'_n\subseteq\mathcal{P}_n, |\mathcal{P}'_n|\leq\Gamma} \sum_{v\in\mathcal{P}'} \hat{m}_{v,n}$
9: **for** $v \in \mathcal{P}_n^{mig}$ **do**
10: $\{\mathcal{P}_n|\forall n \in N\} \leftarrow$ `VNFMigrate`$(v,\ \mathcal{P}_n, \Gamma)$ ▷ Migrate v to another DC
11: **end for**
12: **end if**
13: **end for**
14: $\mathcal{P}_j \leftarrow \mathcal{SP}_{n_1,n_2},\ \forall S \in \mathcal{U},\ \forall j \in S_J : j.source \in \mathcal{P}_{n_1} \wedge j.dest \in \mathcal{P}_{n_2}$
15: $\mathcal{P}_e \leftarrow \{p \mid \forall (n_1, n_2) \in N,\ \forall p \in \mathcal{SP}_{n_1,n_2} : e \in p\},\ \forall e \in E$
16: $\mathcal{M}_{route} \leftarrow$ `ConstructRouting`$(\mathcal{P}_n, \mathcal{P}_j, \mathcal{P}_e)$ ▷ Construct the routing model
17: $\{\hat{x}_j^p|\forall S \in \mathcal{U}, \forall j \in \mathcal{S}_J, \forall p \in \mathcal{P}_j\} \leftarrow$ Solves the integral-relaxed model $\mathcal{M}_{route}$
18: **if** $\hat{x}_j^p = None$ **then** ▷ If relaxed the model is infeasible
19: $E^{irr}, S^{irr} \leftarrow$ `ComputeIIS`$(\mathcal{M}_{route})$ ▷ Compute the irreducible system
20: $\{N_v|\forall S \in \mathcal{U}, \forall v \in S_V\} \leftarrow$ `UpdatePotentialLocs`(E^{irr}, S^{irr})
21: Go back to line 4 ▷ Redo the placement according to the updated N_v in the next iteration
22: **end if**
23: $\hat{x}_j^p \leftarrow 1,\ \forall S \in \mathcal{U},\ \forall j \in S_J : p' = \text{argmax}_{p\in\mathcal{P}_j}\{\hat{x}_j^p\}$ ▷ Select a path for each flow
24: $\mathcal{J}_e = \{j : \forall S \in \mathcal{U},\ \forall j \in S_J, \forall p \in \mathcal{P}_j : e \in p \wedge \hat{x}_j^p = 1\},\ \forall e \in E$
25: **for** $e \in E$ **do** ▷ Check the link capacity and do flow migration if need
26: **if** $\sum_{j\in\mathcal{J}_e} \bar{r}_j^{bwd} + \max_{\mathcal{J}'_e\subset\mathcal{J}_e, |\mathcal{J}'_e|\leq\Gamma} \sum_{j\in\mathcal{J}'_e} \hat{r}_j^{bwd} \leq C_e$ **then**
27: $\mathcal{J}_e, \mathcal{J}_e^{rev} \leftarrow$ Solve (B9) with $c_e = c_e - \max_{\mathcal{J}'_e\subset\mathcal{J}_e, |\mathcal{J}'_e|\leq\Gamma} \sum_{j\in\mathcal{J}'_e} \hat{r}_j^{bwd}$
28: **for** $j \in \mathcal{J}_e^{rev}$ **do**
29: $\{\mathcal{J}_e|\forall e \in E\} \leftarrow$ `FlowMigrate`$(j, \mathcal{J}_e, \mathcal{P}_j, \mathcal{P}_e, \Gamma)$ ▷ Migrate j to another path
30: **if** `FlowMigrate` $fails$ **then** ▷ No feasible migration scheme
31: $n_1 = n,\ j.source \in P_n$ ▷ The DC where $j.source$ is placed
32: $n_2 = n,\ j.dest \in P_n$
33: **if** $j.source \neq ingress \vee |N_{j.source}| \geq |N_{j.dest}| \vee \frac{c_{n_1}}{P_{n_1}} < \frac{c_{n_2}}{P_{n_2}}$ **then**
34: $N_{j.source}.remove(n_1)$
35: **else** $N_{j.dest}.remove(n_2)$
36: **end if**
37: Go back to line 4 ▷ Redo the placement according to the updated N_v
38: **end if**
39: **end for**
40: **end if**
41: **end for**
42: $Break$ ▷ Certainly that the routing is feasible
43: **end while**
44: **return** $(\{\mathcal{P}_n|\forall n \in N\}, \{\mathcal{J}_e|\forall e \in E\})$ ▷ Placement and routing schemess

the iterative-rounding algorithm with capacity-constraint-removal [27] to compute a VNF placement scheme (line 5) whose objective value is *no worst* than the optimal solution of the nominal model, accepting that each substrate node may be *over-allocated by at most twice its capacity*.

Greedy VNF Migration. After obtaining an initial placement which may not be feasible in practice, we perform VNF migration to fix the capacity violations while taking into consideration demand deviations (lines 6 - 13). The crux of this step is to do the migration in a manner that retains as much as possible the solution computed by the iterative rounding algorithm in line 4 so that the change to the objective is kept minimal. Recall that the cost for mapping a virtual node is different from one location to another depending on the values of c_n (per-unit processing) and P_n (per-unit cost) at each substrate node n. The virtual nodes ideally should be assigned to the substrate nodes with the highest values of c_n/P_n, as they would incur the least costs. Those are also substrate nodes most likely to have their capacity violated. We iterate through each substrate node in descending order of their c_n/P_n ratios (line 6). At each node, assuming that the Γ most deviated demands will take their maximum deviations, we compute the total demand and verify if it is less than the node capacity M_n (line 7). If the node capacity is not respected, we migrate some virtual nodes originally assigned there to other substrate nodes with available capacity. More specifically, we first deduct the node capacity by the total deviation of Γ most deviated demands and then solve the following recurrence:

$$opt(c,i) = \begin{cases} 0 & if\ i = 0\, or\, c = 0, \\ opt(c, i-1), & if\ d_i > c, \\ \max\{opt(c - d_i, i-1) + d_i, & otherwise, \\ \quad opt(c, i-1)\}, & \end{cases} \tag{B9}$$

to identify the set of virtual nodes to keep ($\mathcal{P}_n$) as well as those to migrate ($\mathcal{P}_n^{mig}$) so that the remaining *usable capacity is maximally utilized* (line 8). For notational simplicity in (B9), we denote by c the capacity of the node being considered and d_i the number of resource units demanded by the i-th virtual node being placed at that node while $opt(c,\ i)$ is the maximal utilization of the capacity c considering the virtual nodes numbered from 1 to i. For each virtual node $v \in \mathcal{P}_n^{mig}$, we then call `VNFMigrate` to migrate v to a new location, considering the substrate nodes with enough capacity ordered from best to worst with respect to their c_n/P_n ratios (lines 9-11).

Optimality Analysis. Let P_{opt}^{robust} and P^{xGSODL} be the total costs returned by the optimal robust model and our approximation algorithm xGSODL, respectively. Theorem 4 characterizes the worst-case performance of xGSODL. For clarity in the presentation, we use simplified notations such as $\sum_j$ to indicate the summation over all virtual links (excluding those destined to an egress) and $\sum_j^\Gamma$ to indicate the summation over the virtual links whose demands are among the Γ most deviated demands.

Theorem 4. $\frac{P^{xGSODL}}{P_{opt}^{robust}} \leq \frac{1}{2} + \left(\frac{1}{2} + \alpha(\Gamma)\right)\frac{P_w c_b}{P_b c_w}$, *where* $b = \text{argmax}_n\{(c_n/P_n) \mid \forall n \in N\}$, $w = \text{argmin}_n\{(c_n/P_n) \mid \forall n \in N\}$, *and*

$$\alpha(\Gamma) = \frac{\left(\sum_j^\Gamma \frac{\hat{r}_j}{\sigma_j}\right)/\left(\sum_j \frac{\bar{r}_j}{\sigma_j}\right) + \left(\left(\frac{1}{2}\sum_j 1\right) + \Gamma + 1\right)c_w}{\sum_j \bar{r}_j/\sigma_j}. \tag{B10}$$

Proof. Let $P_{nominal}^{opt}$ be the optimal cost computed by the non-robust model that considers only the nominal demands, we know that $P_{opt}^{robust} \geq P_{opt}^{nominal}$. This is rather trivial since on top of the resources allocated to the nominal demand, the robust model has to allocate more resources to account for the demand deviations, which of course incurs additional cost. We also know that the total cost returned by the iterative rounding algorithm for generalized assignment (denoted by P^{GAIR}) considering only the nominal demands is no worse than that returned by the non-robust model, hence $P_{opt}^{robust} \geq P_{opt}^{nominal} \geq P^{GAIR}$. From this, we can deduce the following inequalities:

$$\frac{P^{\text{xGSODL}}}{P^{robust}_{opt}} \leq \frac{P^{\text{xGSODL}}}{P^{nominal}_{opt}} \leq \frac{P^{\text{xGSODL}}}{P^{GAIR}} \tag{B11}$$

$$\leq \frac{\left(\left\lfloor \sum_j \left\lceil \frac{\bar{r}_j}{c_b\sigma_j} \right\rceil /2 \right\rfloor - \sum_j^{\Gamma} \left\lceil \frac{\hat{r}_j}{c_b\sigma_j} \right\rceil\right)P_b + \left(\left\lceil \sum_j \left\lceil \frac{\bar{r}_j}{c_w\sigma_j} \right\rceil /2 \right\rceil + \sum_j^{\Gamma} \left\lceil \frac{\hat{r}_j}{c_w\sigma_j} \right\rceil\right)P_w}{\sum_j \left\lceil \frac{\bar{r}_j}{c_b\sigma_j} \right\rceil P_b} \tag{B12}$$

$$\leq \frac{1}{2} + \frac{\left(\frac{1}{2c_w}\left(\sum_j \frac{\bar{r}_j}{\sigma_j}\right) + 1 + \frac{1}{2}\sum_j 1\right)P_w + \left(\frac{1}{c_w}\left(\sum_j^{\Gamma} \frac{\hat{r}_j}{\sigma_j}\right) + \Gamma\right)P_w}{\sum_j \frac{\bar{r}_j}{\sigma_j}\frac{P_b}{c_b}} \tag{B13}$$

$$= \frac{1}{2} + \frac{P_w c_b}{2c_w P_b} + \frac{P_w c_b}{c_w P_b}\frac{\sum_j^{\Gamma} \frac{\hat{r}_j}{\sigma_j}}{\sum_j \frac{\bar{r}_j}{\sigma_j}} + \frac{P_w c_b}{c_w P_b}\frac{\left(\frac{1}{2}\left(\sum_j 1\right) + \Gamma + 1\right)c_w}{\sum_j \frac{\bar{r}_j}{\sigma_j}} \tag{B14}$$

$$= \frac{1}{2} + \left(\frac{1}{2} + \alpha(\Gamma)\right)\frac{P_w c_b}{P_b c_w}. \tag{B15}$$

The inequality (B12) considers the scenario that causes the largest discrepancy between P^{GAIR} and P^{xGSODL}. The denominator in (B12) expresses the lowest possible cost that could be achieved by P^{GAIR}, where all virtual nodes are assumed to be assigned to the substrate node b which provides the best processing capacity per unit of cost (*i.e.*, $b = \text{argmax}_n\{(c_n/P_n)\,|\,\forall n \in N\}$). The numerator of (B12) expresses the total cost as computed by xGSODL considering the worst-case scenario, where virtual nodes decided to be migrated by solving (B9) are all assigned to the substrate node w with the least economical cost per unit of capacity (*i.e.*, $w = \text{argmin}_n\{(c_n/P_n)\,|\,\forall n \in N\}$). Specifically, the first term in (B12)'s numerator expresses the total cost for the remaining virtual nodes at b that are not migrated, while the second term is the total cost of the virtual nodes migrated to w from b. □

B. Routing

VNF Routing as Packing. Given the placement of virtual nodes, our task then is to compute a routing scheme that satisfies capacity constraints and respects the deadline of each SFC. Without even considering the deadline constraint, the problem of finding integer flows that satisfy all capacity constraints is already NP-complete. By taking advantage of the low path diversity exhibited by inter-datacenter networks, we pre-compute candidate paths (as done in [28]) by finding the K-shortest paths in terms of propagation delays for each substrate node pair (line 2) and reformulate the problem as a *packing* problem whose structure can be exploited to simplify the solution. Specifically, let $\mathcal{P}_j$ denote the set of candidate paths for flow j. If $j.source$ and $j.dest$ are mapped to n_1 and n_2, respectively, then $\mathcal{P}_j$ is given by $\mathcal{SP}_{n_1,n_2}$, the set of K-shortest paths precomputed for node pair (n_1, n_2) (line 14). We further have the set $\mathcal{P}_e$ which denotes the set of paths routed through link e (line 15). Given this information, the routing then can be modeled as shown in Formulation 3 (line 16). The output of the model is the path-selection decision x_j^p, a binary variable indicating whether to select path p to route flow j.

While the objective is only to compute a feasible routing scheme, we augment the model with a dummy objective to force it to always compute *extreme point solutions*, which *mostly* consist of 0's and 1's. After having the routing model constructed, we relax the integral constraints and solve the relaxed model with the nominal demands taken as the input (line 17). Let $\tilde{x}_j^p$ denote the relaxed solution of x_j^p. Theorem 5 presents the preconditions on the model so that there will always be some 0's and 1's in the computed solutions.

Formulation 3 $\mathcal{M}_{route}$

$$Minimize \sum_{S\in\mathcal{U}}\sum_{j\in S_J}\sum_{p\in\mathcal{P}_j} x_j^p d_p^{prp} \tag{B16}$$

$$\sum_{p\in\mathcal{P}_j} x_j^p = 1,\ \ \forall S \in \mathcal{U}, \forall j \in \mathcal{S}_J, \tag{B17}$$

$$\sum_{p\in\mathcal{P}_e}\sum_{S\in\mathcal{U}}\sum_{j\in S_J} x_j^p < c_e,\ \ \forall e \in E, \tag{B18}$$

$$\sum_{j\in\mathcal{S}_j}\sum_{p\in\mathcal{P}_j} x_j^p d_p^{prp} \leq \theta_S,\ \ \forall S \in \mathcal{U}, \tag{B19}$$

$$x_j^p \in \{0,1\}, \quad \forall S \in \mathcal{U}, \forall j \in S_J . \forall p \in \mathcal{P}_j$$

Theorem 5. *If* $\sum_{S\in\mathcal{U}}|S_J|-1>|E'|$, *then* $\exists S \in \mathcal{U}, j \in S_J, p \in \mathcal{P}_j, \tilde{x}_j^p \in \{0,1\}$ *where* $E' = \{e \in \mathcal{P}_j \,|\, \forall S \in \mathcal{U}, \forall j \in S_J, \forall e \in E\}$.

Proof. Consider proving its contra-positive by assuming that there does not exist any relaxed variable that has the value of 0 or 1, i.e., $\forall S \in \mathcal{U}, j \in S_J, p \in \mathcal{P}_j, \tilde{x}_j^p \notin \{0,1\}$. We have $\sum_{S\in\mathcal{U}}|S_J|$ virtual links, each with a set of $\mathcal{P}_j$ candidate paths, for a total of $\sum_{S\in\mathcal{U}}\sum_{j\in S_J}|\mathcal{P}_j|$ variables. If there are no variables with values of 0 or 1, then by the property of extreme point solutions, we must have at least $\sum_{S\in\mathcal{U}} 2|S_J|$ *independent tight constraints*. However, by the model definition, we have at most $\sum_{S\in\mathcal{U}}|S_J|+|E|+|\mathcal{U}|$ constraints. Further, note that depending on the number of precomputed paths K, a substrate link e may not be included in any set of paths $\mathcal{P}_j$ for any flow j. So, excluding those links, there can only be at most $|E'|$ active link capacity constraints (B18). Thus, we have $2\sum_{S\in\mathcal{U}}|S_J| \leq \sum_{S\in\mathcal{U}}|S_J|+|E'|+|\mathcal{U}| \leq \sum_{S\in\mathcal{U}}|S_J|+|E|+|\mathcal{U}| \Leftrightarrow \sum_{S\in\mathcal{U}}|S_J| \leq |E'|+|\mathcal{U}| \Leftrightarrow \sum_{S\in\mathcal{U}}|S_J|-1 \leq |E'|$. □

In practice, $|E'|$ is typically smaller than $\sum_{S\in\mathcal{U}}|S_J|-1$ as there are many more flows in the system compared to the size of the inter-datacenter network. As we will later show in the evaluation, considering a thousand instances of the problem, the relaxed solution is fully integral in more than 90% of the time, and even in the case the relaxed solution is not completely integral, less than 5% of the variables are indeed fractional. Taking advantage of the fact that the virtual link assignment rarely splits a flows over multiple paths, instead of solving the model iteratively, we solve it once then simply select the path p with the largest value of $\tilde{x}_j^p$ to route j (line 23).

Routing Feasibility. When $\Gamma = 0$ and all $\tilde{x}_j^p$ are already fully integral, as we are assured to obtain a feasible routing scheme, the link capacity check in lines 25-40 can be skipped. For the general case when $\Gamma \geq 0$ and some $\tilde{x}_j^p$ are fractional, we iterate through each link and verify if the total bandwidth demands (considering both the nominal and uncertain demands) exceed the link capacity. In the same manner that the node capacity violation is handled, for any link with insufficient capacity, we solve the recurrence (B12) to identify a minimal amount of flows to migrate to satisfy the substrate link capacity (line 27) and call `FlowMigrate` to migrate the corresponding virtual links to different paths which do not use

this substrate link (line 28). For each virtual link j to migrate, `FlowMigrate` iteratively checks through the set of paths in $\mathcal{P}_j$ and assigns j to the first path with sufficient capacity whose propagation delay does not violate the deadline of the SFC that j belongs to. It is also important to point out that given that there is a path that can satisfy the deadline constraint, the deadline constraint cannot be violated as the result of rounding unless the link capacity is also violated since if there is capacity available at the shortest path, there is no reason for the model to split a flow into multiple paths.

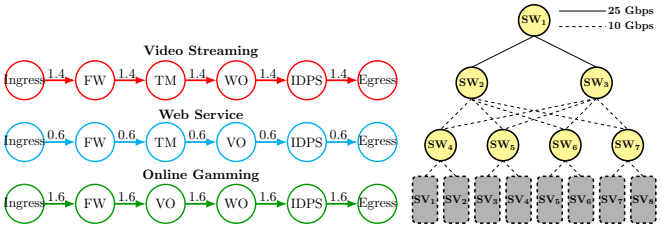
Placement Routing Dependency. Given a placement scheme, there may not be any feasible routing scheme that can meet the service deadlines and link capacity constraints. In such cases, there may be no feasible fractional routing scheme (line 18), or there is, but it has no feasible rounded solution (line 30). In the case that the relaxed model is infeasible, we compute its irreducible inconsistent subsystem (IIS), which consists of a list of constraints and variables that cause the model to be infeasible (line 19). If the infeasibility is caused by a link capacity violation, we select the smallest flow j which competes for the capacity on that link and relocate either $j.source$ or $j.dest$ to a different substrate node by updating their list of possible locations N_v (line 20) and redo the placement (line 21). If the IIS indicates that a deadline constraint is too tight for an SFC given a placement, we select the virtual link j along that SFC whose shortest path in $\mathcal{P}_j$ has the smallest propagation delay to do the relocation. Likewise, for the case that we cannot find a feasible migration for a flow j to fix the capacity violation as the result of rounding (lines 30-37), we rerun the placement algorithm with the list of possible locations N_v for $j.source$ or $j.dest$ updated to omit the current location. For all scenarios above, the decisions on whether to select $j.source$ or $j.dest$ to relocate are based on the following precedence rules: (1) neither an ingress or egress, (2) has a bigger set of N_v, and (3) currently placed at a less cost-efficient location (*i.e.*, lower C_n/P_n).

VII. Performance Evaluation

We conduct extensive simulations to demonstrate the performance of our proposed algorithms for co-located (SODL and xSODL) and geo-distributed SFC orchestration (xGSODL), in terms of their ability to cope with demand fluctuations, scalability and relative performance against other algorithms. All algorithms are implemented in Python 2.7 and run on an Intel Core i7-3720QM@2.6GHz machine with 16 GB RAM. We utilize the Python interface of the Gurobi 8.1 optimizer [24] to implement and solve the optimization models.

A. SODL Evaluation

Simulation Settings. We consider orchestrating 3 SFCs, whose compositions and nominal service demands (in Gbps) are shown in Fig. 3(a) (these SFCs are described in [29]). The 12 VNF requests in the service chains belong to five different types, each with its own processing coefficient. We assume that 3 licenses for each VNF type are available. By default, the deadlines for all SFCs are set to 0.2 ms. These deadlines



(a) Set of SFCs. (b) NFV infrastructure.

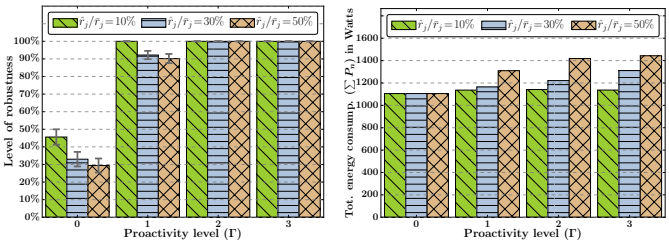
Fig. 3: Exact model evaluation settings.

are strictly enforced by SODL as λ is set to be very large. The NFVI is a Clos data center topology, as illustrated in Fig. 3(b), with 8 servers, 7 switches and 36 directed links which have negligible propagation delays. The core switch SW_1 is set to be both the ingress and egress location of all SFCs. Table III summarizes the configurations for servers, switches and VNFs.

TABLE III: Settings for servers, switches and VNFs (from [30–32]).

Server configurations (4 classes)				
No. of cores (M_n)	4 cores	6 cores	8 cores	10 cores
Raw core capacity (c_n^{cpu})	1.0 Gbps	1.0 Gbps	1.0 Gbps	1.0 Gbps
Idle power (P_n^{active})	70 Watts	75 Watts	80 Watts	100 Watts
Max power (P_n^{max})	200 Watts	250 Watts	300 Watts	350 Watts
Switch configurations (1 class)				
Switching capacity (c_n^{bwd})	120.0 Gbps			
Idle power (P_n^{active})	30 Watts			
Max power (P_n^{max})	60 Watts			
VNF configurations (5 types)				
VNF f	Abbr.	Processing coef. (σ_f)	No. of licenses (K_f)	
Traffic Monitor	TM	1.0	3	
Fire Wall	FW	0.9	3	
Intrusion Detection System	IDS	0.7	3	
WAN Optimizer	WO	0.6	3	
Video Optimizer	VO	0.6	3	

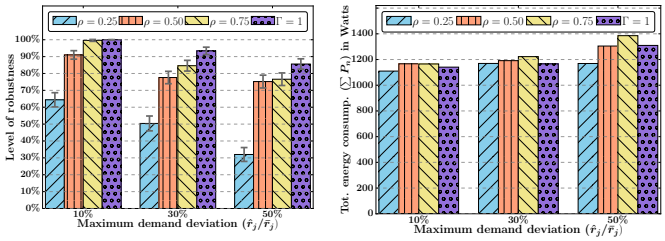
Effect of Demand Fluctuations. The goal of this experiment



(a) Effect of demand fluctuations. (b) Price of robustness.

Fig. 4: Performance of SODL under different proactivity level.

is to show: (i) the ability of SODL to proactively cope with demand fluctuations, and (ii) the effect of the proactivity of SODL on the service orchestration cost. We show the results for different values of Γ, which is the parameter that controls the proactivity of the algorithm. We consider three demand profiles with relative maximum deviations of $\hat{r}_j/\bar{r}_j = 10\%, 30\%$, and 50%. For each demand profile, first we run SODL to compute the corresponding orchestration. Then, we randomly generate 500 demand vectors (consistent with the demand profile), where each vector consists of three demands, one for each deployed SFC. For each demand vector, we check whether the demands can be satisfied by the computed orchestration scheme. This allows us to compute the percentage of feasible (infeasible) demands,



(a) Relative protection levels. (b) Relative energy cost.

Fig. 5: Performance of SODL against the baseline algorithm

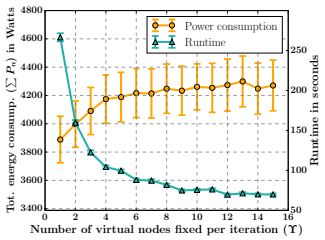


Fig. 6: Runtime-energy trade-off in xSODL.

which indicates how proactive (reactive) is the orchestration algorithm. Fig. 4(a) shows the percentage of feasible demands (with 95% confidence intervals) as the proactivity level increases. We observe that when the proactivity level is 0, frequent reprovisioning is required. However, as the proactivity level increases, the percentage of feasible demands increases significantly. In particular, setting Γ to be as low as one makes the computed orchestration immune to demand fluctuations in 90% of cases. Fig. 4(b) plots the energy cost of service orchestration against the proactivity level Γ for the three demand profiles. As evident from the figure, the energy consumption increases monotonically with Γ. This is expected since by increasing Γ, the model becomes more proactive and provisions more network resources to cope with higher future demand fluctuations, thus increasing the energy consumption.

Comparision with a Baseline Approach. We compare SODL with a *baseline* algorithm, which is constructed by adding a *safety margin* to each SFC's nominal demand to account for future demand fluctuations. Specifically, if the demand profile of flow j is given by $[\bar{r}_j - \hat{r}_j, \bar{r}_j + \hat{r}_j]$, then the baseline algorithm assigns the demand $\bar{r}_j + \rho\hat{r}_j$ to j, for some $0 \leq \rho \leq 1$. Specifically, setting ρ to 1 results in a fully proactive orchestration, while setting ρ to 0 leads to an orchestration that is susceptible to any demand fluctuation. The baseline model is solved by setting $\Gamma = 0$ in our exact model and using deterministic service demands computed with respect to various values of ρ. Figs. 5(a) and 5(b), respectively, compare the robustness and energy cost of the baseline using three values for ρ (*i.e.*, 0.25, 0.50 and 0.75) against SODL with $\Gamma = 1$. As in the previous experiment, to measure robustness, we use the computed orchestration scheme to accommodate 500 randomly generated demands and record the percentage of feasible demands. The results shown in the figures reveal that SODL provides better protection against demand fluctuations while at the same time consuming less energy than the baseline algorithm for all considered values of relative maximum deviation $\hat{r}_j/\bar{r}_j$.

B. xSODL Evaluation

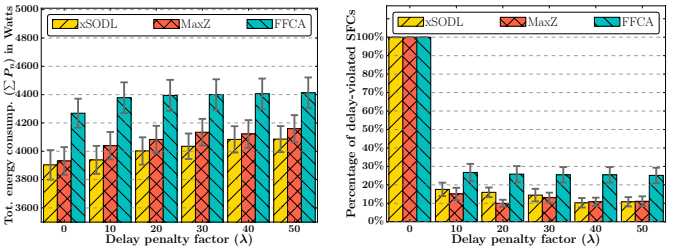
Simulation Settings. To assess the performance of xSODL, we consider a scaled-up version of the scenario considered for the evaluation of SODL. Specifically, we triple the number of SFCs, increasing the number of VNF requests from 12 to 36. The number of licenses for each VNF is also increased to 10. The nominal demands here are randomly generated over the range $[0.5, 2.5]$ Gbps. The NFVI is still a Clos datacenter topology as shown in Fig. 3(b), but has a larger capacity with 24 servers, 21 switches and 132 directed links. All other settings remain the same. To solve larger problem instances, one can increase Υ and/or decrease the SFC batch size to reduce the number of simultaneous SFCs that are placed by the algorithm.

Runtime-Energy Trade-off. In this experiment, we run xSODL on randomly generated demand profiles and record the average energy consumption and runtime for different values of Υ. Recall that Υ is the number of VNFs fixed in each iteration of the model. The results with 95 % confidence intervals are reported in Fig. 6. As expected, fixing more virtual nodes per iteration reduces the time it takes to run the model, thereby improving the runtime of the algorithm. It can be seen from the runtime plot that as we double Υ, the runtime reduces almost by half. Therefore, depending on the problem scale, Υ can be set to allow the algorithm to run in a reasonable time. Though this benefit comes at a price. As Υ increases, the likelihood of virtual nodes being assigned to unfavorable locations, which incur higher energy cost, increases since more fractional assignment solutions are selected to be rounded. This behavior is reflected in the energy plot shown in Fig. 6.

Comparison with Existing Algorithms. We have implemented the following algorithms for comparison with xSODL:

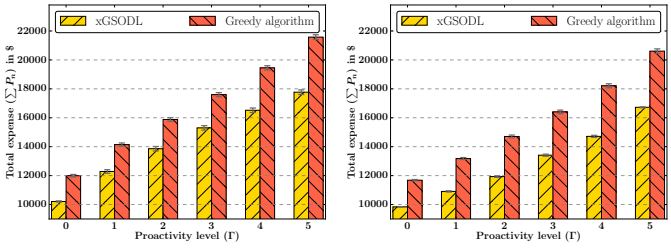
- **First Fit Clustering Allocation** (FFCA) [7]: In this algorithm, servers connecting to the same switch form a cluster. FFCA tries to allocate all VNF requests on the same SFC to servers within a cluster, with the server that has the most amount of resources being considered first.
- **MaxZ** [8]: Similar to xSODL, MaxZ uses iterative rounding for VNF placement. Different from xSODL, however, they do not prevent the model from fractionally assigning VNF requests to servers that do not have enough capacities in the next run. After each run of the relaxed model, MaxZ selects the largest fractional assignment that meets the capacity requirement to fix the placement.

We note that both FFCA and MaxZ are only general approaches for VNF mapping, they do not consider routing or CPU allocation. As such, we only implement them as alternative approaches for picking a location for mapping a VNF request, while using the same procedures as in xSODL for the CPU allocation and traffic routing. Fig. 7(a) and 7(b), respectively, show the effect of increasing λ on the energy provisioning cost and the percentage of SFCs that do not meet the deadline. The 95% confidence intervals shown in



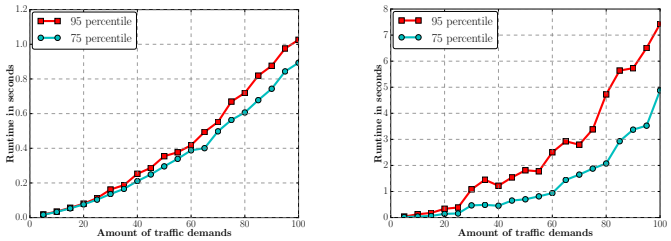
(a) Relative cost. (b) Relative delay performance.

Fig. 7: Comparing xSODL with existing algorithms.



(a) $|N_v| = 10$ (b) $|N_v| = 20$

Fig. 8: Comparison of xGSODL and Greedy algorithms.



(a) $\theta_S \in \mathcal{U}[500, 600]$ (b) $\theta_S \in \mathcal{U}[450, 550]$

Fig. 9: xGSODL's runtime as the level of resource competition increases.

the figures are calculated based on 100 runs of the algorithms on randomly generated demand profiles. We observe that xSODL performs consistently better than the other two approaches in terms of energy usage. However, in terms of the quality of service (*i.e.*, indicated by the percentage of deadline-violated SFCs), while xSODL consistently outperforms FFCA, xSODL only performs better than MaxZ when λ is set to be sufficiently large. When λ is small, the deadline violation does not significantly affect the objective, and since MaxZ is less energy-efficient and provisions more resources than xSODL, it is likely to cause fewer deadline violations. It is also worth noting that setting λ to zero forces the algorithm to provision just enough resources to meet the service demands, which leads to 100% deadline violations.

C. xGSODL Evaluation

Simulation Settings. We consider the 28-nodes US Long-haul [33] to be our network infrastructure. Given the distance information between nodes in [33], we set the propagation delay on each link to be proportional to its length, resulting in delays between 15 and 35 ms. The traffic demands are assumed to deviate by at most 20% of their nominal values, which are uniformly randomly chosen between 8-12 Gbps. With the capacity of each link randomly set to be between 35 and 60 Gbps, at most 2 to 3 flows can be routed through a single path. For an SFC of length 5 as considered throughout this evaluation, its end-to-end delay is expected to be somewhere between 300-800 ms. By setting default service deadlines to be 500-600 ms, we are able to create problem instances where a placement scheme results in no feasible routing scheme, forcing xGSODL to run multiple iterations. Each substrate node is assumed to have 8-12 processing units available for VNF services, with each unit providing 4-8 Gbps processing capacity while incurring a cost of \$200-\$600. We further assume that each VNF has a processing coefficient between 0.5 and 1.0 and is supported by only a subset of substrate nodes randomly selected from N (*i.e.*, $N_v \subset N$).

Comparisons with a Greedy approach. In this experiment, we evaluate the performance of xGSODL against a greedy algorithm. The greedy algorithm sorts the substrate nodes in descending order of their cost efficiency (*i.e.*, C_n/P_n) and maps each virtual node v in a first-fit manner to the most cost-effective substrate node as allowed by its N_v. For each setting used in this experiment, we run both algorithms on 500 problem instances, each of which involves the deployment of 15 randomly generated SFCs. Fig. 8 reports the average costs (with 95% confidence intervals) of the placement schemes computed by the two algorithms as Γ increases. In both scenarios when $|N_v| = 10$ and $|N_v| = 20$, we can see that the greedy algorithm costs 20% to 40% more than xGSODL. Note that by formulating VNF placement as an instance of generalized assignment, xGSODL takes into consideration the relative capacities (*i.e.*, relative M_n) of the substrate nodes when performing the placement, whereas the greedy algorithm just greedily packs as many virtual nodes as possible in the substrate nodes with the least costs without considering their capacities. As such, the greedy algorithm tends to leave many substrate nodes with a small gap of usable capacity unutilized, leading to a higher overall cost. Compared to when $|N_v| = 10$, we also note that the costs given by both algorithms are cheaper when $|N_v| = 20$ as a result of the algorithm having more freedom in choosing where to map the virtual nodes.

Resource Competition versus Runtime. The runtime of xGSODL is not determined primarily by the size of the network but the level of resource competition, or more specifically, the user traffic demands relative to the amount of available resources. Fig. 9 plots the 95th and 75th percentiles of xGSODL runtime as a function of traffic demands considering two different settings for θ_S. The percentiles are computed over 200 runs of xGSODL on randomly generated demand profiles. When $\theta_S \in \mathcal{U}[500, 600]$, Fig. 9(a) shows that xGSODL runtime increases almost linearly with the amount of traffic demands. Specifically, for any problem instance with less than 100 traffic demands, we can expect that in 95% of the cases, xGSODL will take no more than 1 second to run. The effect of resource competition on xGSODL runtime becomes more apparent in Fig. 9(b) when θ_S is set to be more stringent (*i.e.*, $\theta_S \in \mathcal{U}[450, 550]$). As there is less flexibility in terms of delay, it is more likely that a computed

TABLE IV: Effects of different parameters on the routing solutions.

$\lvert N_v\rvert$	$\sum_{S\in\mathcal{U}}\lvert S_J\rvert$	$K=2$			$K=4$			$K=7$			$K=10$		
		η_1	η_2	η_3	η_1	η_2	η_3	η_1	η_2	η_3	η_1	η_2	η_3
1	25	99%	95%	2.0	97%	97%	2.0	94%	98%	2.0	93%	99%	2.0
	50	84%	97%	2.1	82%	99%	2.1	69%	99%	2.2	72%	99%	2.3
2	25	99%	90%	4.0	93%	97%	2.3	88%	98%	2.0	87%	99%	2.0
	50	65%	97%	2.5	63%	98%	2.5	60%	99%	2.6	60%	99%	2.6
3	25	100%	-	0	93%	97%	2.0	92%	98%	2.0	93%	99%	2.0
	50	57%	97%	2.4	65%	98.0%	2.8	62%	99%	2.6	64%	99%	2.6
5	25	99%	94%	2.0	95%	96.8%	2.4	89%	98%	2.2	89%	99%	2.0
	50	57%	97%	2.5	52%	98.1%	3.0	49%	99%	2.9	50%	99%	2.9
10	25	99%	94%	2.0	92%	97%	2.3	87%	98%	2.3	89%	98%	2.4
	50	57%	97%	2.5	54%	98%	2.7	51%	98%	3.1	56%	99%	3.0
20	25	97%	94%	2.0	83%	97%	2.1	83%	97%	2.0	83%	98%	2.2
	50	62%	96%	2.4	30%	98%	2.7	29%	99%	2.8	30%	99%	2.7
25	25	100%	-	0	84%	97%	2.1	82%	98%	2.0	85%	98%	2.0
	50	59%	97.0%	2.3	24%	98%	2.5	18%	99%	2.5	18%	99%	2.8

placement scheme will not have a feasible routing solution. xGSODL therefore has to redo the placement stage more often, resulting in a longer runtime. A larger gap between 95th and 75th percentiles in Fig. 9(b) indicates a larger variation in xGSODL runtime as the deadline becomes more strict.

Routing Integrality. This experiment studies the effects of parameters such as the degree of support for each virtual node $|N_v|$, the total traffic demand $\sum_{S\in\mathcal{U}}|S_J|$, and the number of precomputed paths K on the integrality of the relaxed M_{route}. For each setting of these parameters, we make 200 runs of xGSODL on randomly generated demand vectors and record the following information: η_1 - the percentage of runs in which the relaxed solutions are all integral, η_2 - the average percentage of variables that are not integral, and η_3 - the average number of non-integral solutions. All collected data are reported in Table IV. As $|N_v|$ becomes bigger, we observe that η_1 decreases for all considered traffic loads and numbers of precomputed paths. When $|N_v|$ increases, constraint (B17) in M_{route} will have more variables with non-zero coefficients, making it less likely a linear combination of other constraints. As such, there are less dependent constraints in M_{route}, and consequently, less chance that the relaxed solutions will be fully integral. And even for those relaxed solutions that are not fully integral, records for η_2 and η_3 indicate a vast majority of the relaxed variables are still integers. We further observe that η_1 has a tendency to decrease when K increases. This is because with a bigger set of paths considered by each flow, more link capacity constraints will be active (*i.e.*, bigger $|E'|$), thus increasing the upper bound on the number of active variables and hence the likelihood of fractional assignments. The data also suggests a strong influence of the traffic load on the integrality of the relaxed solutions. Note that with a smaller set of traffic demands, there will be less competition for resources on each path, allowing the relaxed model in the majority of cases to fully assign a flow to the shortest path.

VIII. Related Work

We review relevant works in three categories, as follows. **Reactive Algorithms.** These works, commonly consider *deterministic* service demands (nominal or worst-case) when orchestrating SFCs offline [34]. In [35], the authors present a graph-matching based solution for the service placement problem in distributed fog computing, but ignore traffic routing. The works [8], [19] investigate sharing VNF instances between multiple flows and compute the optimal split of traffic. However, it is unclear how the number of instances can be pre-determined, independent of the demand. While reactive, [36] aims at minimizing the migration frequencies, thereby avoiding constant disruption of services. In this work, we try to avoid VNF migration by proactively accounting for future fluctuations in service demands while keeping the cost minimal. The work [37] provides an energy and deadline-aware SFC orchestration mechanism in datacenters, though it neglects the load-dependent delay components (*e.g.*, queueing delay).

Online Algorithms. Online algorithms, assume no information about future service demands and perform the orchestration dynamically to satisfy any possible future service demands without frequent resource reprovisioning. The authors in [15], [38] develop online SFC orchestration algorithms based on the solutions to the ski-rental and weight matching problems. The work [39] proposes the decoupling of routing and placement decisions which can significantly simplify the design of an online algorithm. The work [40] investigates the joint placement of service chains and steering of traffic through them over a 5G multi-technology edge network, assuming that service demands are constant. Online gradient descent is used in [13] to obtain estimates of future traffic demands which allow to compute SFC orchestrations that are better prepared for changes in demands. This approach is complementary to our work, as the online predictions can be used to estimate demand deviations that are used in our model. Considering only the placement of VNFs, the work [41] applies the concept of modularly varying goals to design an online genetic algorithm that mimics biological evolution to provide better adaptability to demand variations.

Robust Algorithms. A few works have considered demand uncertainty when orchestrating SFCs. The work in [17] formulates SFC orchestration as a robust optimization problem with the objective of maximizing the NFVI operator's profit. More relevant to our work is [16], in which the authors propose a robust orchestration model that aims at minimizing the energy consumption of the NFVI. Their model, however, is computationally expensive to solve. To speed up solving this model, the heuristic approach in [7] solves the placement, routing and resource allocation sub-problems separately, which can result in sub-optimal solutions. In a follow-up work [14], the authors use a robust binary optimization problem, which can be solved efficiently. This work, however, considers a simplified delay model, where a deadline is considered for each pair of consecutive VNFs in a hop-by-hop manner. The work [42] assumes that the demands are normally distributed and uses capacity chance-constraints to model the uncertain resources in the geo-distributed SFC orchestration problem. This work, however, only considers fixed propagation delays.

IX. Conclusion

This paper studied the co-located and geo-distributed SFC orchestration problems. We formulated the problems as robust optimization problems and developed exact and

approximate algorithms to solve them. By leveraging uncertain demand information, our algorithms are able to compute service orchestration solutions that proactively account for fluctuations in service demands. The simulation results revealed that our algorithms are cost-effective and work well under general service demands without requiring frequent resource reprovisioning. An interesting future study would be to design algorithms for other system objectives based on the techniques developed in this paper and evaluate their effects.

REFERENCES

[1] M. Chiosi *et al.*, "Network functions virtualization: An introduction, benefits, enablers, challenges & call for action," *ETSI White Paper*, 2012.
[2] ETSI, "NFV management and orchestration," *ETSI GS NFV-MAN*, 2014.
[3] H. Ballani *et al.*, "Towards predictable datacenter networks," in *ACM SIGCOMM Comput. Commun. Rev.*, vol. 41, no. 4, 2011.
[4] J. Duan *et al.*, "Dynamic Scaling of Virtualized, Distributed Service Chains: A Case Study of IMS," *IEEE J. Sel. Areas Commun.*, vol. 35, no. 11, 2017.
[5] B. Yi *et al.*, "A comprehensive survey of network function virtualization," *Computer Networks*, vol. 133, 2018.
[6] D. Bertsimas, D. B. Brown, and C. Caramanis, "Theory and applications of robust optimization," *SIAM Review*, vol. 53, no. 3, 2011.
[7] A. Marotta *et al.*, "A fast robust optimization-based heuristic for the deployment of green virtual network functions," *Journal of Network and Computer Applications*, vol. 95, 2017.
[8] S. Agarwal *et al.*, "Joint VNF placement and CPU allocation in 5G," in *IEEE INFOCOM*, 2018.
[9] Q. M. Nguyen, "Deadline-aware service function orchestration under demand uncertainty," Master's thesis, Science, 2020. [Online]. Available: https://prism.ucalgary.ca/bitstream/handle/1880/111528/
[10] J. Halpern and C. Pignataro, "Service Function Chaining (SFC) Architecture," Internet Requests for Comments, RFC Editor, RFC 7665, October 2015. [Online]. Available: https://tools.ietf.org/html/rfc7665
[11] R. Doriguzzi-Corin *et al.*, "Dynamic and application-aware provisioning of chained virtual security network functions," *IEEE Trans. Netw. Service Manag.*, vol. 17, no. 1, pp. 294–307, 2020.
[12] B. Yi, X. Wang, and M. Huang, "A generalized VNF sharing approach for service scheduling," *IEEE Commun. Lett.*, vol. 22, no. 1, 2018.
[13] X. Zhang *et al.*, "Proactive VNF provisioning with multi-timescale cloud resources: Fusing online learning and online optimization," in *IEEE INFOCOM*, 2017.
[14] A. Marotta *et al.*, "On the energy cost of robustness for green virtual network function placement in 5G virtualized infrastructures," *Computer Networks*, vol. 125, 2017.
[15] X. Wang *et al.*, "Online VNF scaling in datacenters," in *IEEE CLOUD*, 2016.
[16] A. Marotta and A. Kassler, "A power efficient and robust virtual network functions placement problem," in *IEEE ITC*, 2016.
[17] V. S. Reddy, A. Baumgartner, and T. Bauschert, "Robust embedding of VNF/service chains with delay bounds," *IEEE NFV-SDN*, 2016.
[18] D. T. Nguyen *et al.*, "Real-time optimized nfv architecture for internetworking webrtc and ims," in *IEEE ITNSP Sym. Net.*, 2016.
[19] S. Agarwal *et al.*, "VNF Placement and Resource Allocation for the Support of Vertical Services in 5G Networks," *IEEE/ACM Trans. Netw.*, vol. 27, no. 1, 2019.
[20] S. Albers, "On energy conservation in data centers," *ACM Trans. Parallel Comput.*, vol. 6, no. 3, Nov. 2019. [Online]. Available: https://doi.org/10.1145/3364210
[21] M. Dayarathna, Y. Wen, and R. Fan, "Data center energy consumption modeling: A survey," *IEEE Commun. Surveys Tuts.*, vol. 18, no. 1, 2016.
[22] N. F. V. NFV and U. Cases, "ETSI GS NFV 001 V1. 1.1 (2013-10)."
[23] Y. Jia *et al.*, "Online scaling of NFV service chains across geo-distributed datacenters," *IEEE/ACM Trans. Netw.*, vol. 26, no. 2, pp. 699–710, 2018.
[24] Gurobi. (2018). [Online]. Available: http://www.gurobi.com/
[25] S. Zymler, D. Kuhn, and B. Rustem, "Distributionally robust joint chance constraints with second-order moment information," *Mathematical Programming*, vol. 137, no. 1-2, 2013.
[26] S. Patterson, N. McGlohon, and K. Dyagilev, "Optimal k-leader selection for coherence and convergence rate in one-dimensional networks," *IEEE Trans. Control Netw. Syst.*, vol. 4, no. 3, 2017.
[27] L. C. Lau, R. Ravi, and M. Singh, *Iterative methods in combinatorial optimization*. Cambridge University Press, 2011, vol. 46.
[28] C.-Y. Hong *et al.*, "Achieving high utilization with software-driven WAN," in *ACM SIGCOMM Comput. Commun. Rev.*, vol. 43, no. 4, 2013.
[29] N. Huin, B. Jaumard, and F. Giroire, "Optimal network service chain provisioning," *IEEE/ACM Trans. Netw.*, 2018.
[30] Cisco. (2018) Power Calculator. [Online]. Available: http://ucspowercalc.cisco.com
[31] Aruba. (2018) 3810 Switch Series. [Online]. Available: https://www.arubanetworks.com/assets/ds/DS_3810SwitchSeries.pdf
[32] J. Martins *et al.*, "ClickOS and the art of network function virtualization," in *USENIX NSDI*, 2014.
[33] P. Datta Choudhury, N. Agarwal, and T. De, "Spectrum and splitter utilization efficient traffic grooming routing and spectrum assignment in elastic optical networks," in *IEEE SPICES*, 2017.
[34] J. G. Herrera and J. F. Botero, "Resource allocation in NFV: A comprehensive survey," *IEEE Trans. Netw. Service Manag.*, vol. 13, no. 3, 2016.
[35] F. Chiti *et al.*, "Virtual functions placement with time constraints in fog computing: A matching theory perspective," *IEEE Trans. Netw. Service Manag.*, vol. 16, no. 3, 2019.
[36] D. Harutyunyan *et al.*, "Latency-Aware Service Function Chain Placement in 5G Mobile Networks," in *IEEE NetSoft*, 2019.
[37] M. M. Tajiki *et al.*, "Joint energy efficient and QoS-aware path allocation and VNF placement for service function chaining," *IEEE Trans. Netw. Service Manag.*, vol. 16, no. 1, 2019.
[38] X. Wang *et al.*, "Online learning-assisted VNF service chain scaling with network uncertainties," in *IEEE CLOUD*, 2017.
[39] B. Zhang *et al.*, "Toward online virtual network function placement in software defined networks," in *IEEE/ACM IWQoS*, 2016.
[40] N. Akhtar *et al.*, "Virtual function placement and traffic steering over 5g multi-technology networks," in *IEEE NetSoft*, 2018.
[41] M. Otokura *et al.*, "Evolvable virtual network function placement method: Mechanism and performance evaluation," *IEEE Trans. Netw. Service Manag.*, vol. 16, no. 1, 2019.
[42] D. Chemodanov, P. Calyam, and F. Esposito, "A near optimal reliable composition approach for geo-distributed latency-sensitive service chains," in *IEEE INFOCOM*, 2019.