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.

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 proactiveness-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 $G(N, E)$, 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 providing 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 $r_{v,bud}^e$ denote the aggregate rate of traffic on link $e \in E$, which should not be greater than the link capacity $c_{v,bud}^e$. 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 $\sigma_f$, 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 $\sigma_f$ close to 1. On the other hand, processing-intensive VNFs such as DPI have smaller values of $\sigma_f$.

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 $\mathcal{G} = (S_V, S_L)$, where $S_V$ and $S_L$ 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 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 $r_{v,bud}^j$. We assume single-path routing, where each virtual link is 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 $N = N_{\text{switch}} \cup N_{\text{server}}$, in which $N_{\text{switch}}$ and $N_{\text{server}}$ denote the sets of switches and servers, respectively. We assume that only server nodes are capable of hosting VNFs (i.e., $\forall S \in \mathcal{U}, \forall v \in S_V, N_v = N_{\text{server}}$).

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 $r_{v,bud}^f$ for VNF instance $f$ on server $n$, we allocate $i_{f,n}$ 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 $d_{f,n}^{\text{proc}}$ of VNF $f$ on server $n$ is determined by the allocated processing capacity $c_{f,n}$ and its processing rate requirement $r_{f,n}$. 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 $d_{l,e}^{\text{prop}}$ and a queuing delay $d_{l,e}^{\text{bwd}}$. The propagation delay is always fixed. The queuing delay, on the other hand, is a function of the traffic load on the link. We estimate the queuing delay by the response time of an $M/G/1$ 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 $P_n^{\text{actv}}$ 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 $P_{n,\text{max}}$ 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 queuing 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 $d_{l,e}^{\text{prop}}$ 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 – $o_v,n$, CPU allocation – $m_{f,n}$ and routing – $l_{f,e}$ 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:

$$
P_n = \begin{cases} 
0, & \text{if } m_n \leq P_n^{\text{actv}}, \forall n \in N_{\text{server}}, \\
\frac{\sum_{f \in F} m_{f,n}}{M_n} (P_{n,\text{max}} - P_n^{\text{actv}}), & \text{if } m_n > P_n^{\text{actv}}, \forall n \in N_{\text{server}}, \\
\frac{\sum_{v \in V} \sum_{e \in E_{\text{bwd}}} d_{l,e}^{\text{bwd}} u_{v,e}}{P_n^{\text{actv}}} (P_{n,\text{max}} - P_n^{\text{actv}}), & \forall n \in N_{\text{switch}}. 
\end{cases}
$$

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 $o_v,n$. Constraint (A3) computes the processing capacity $c_{f,n}$ of VNF $f$ on server $n$ based on the number of allocated CPU cores $m_{f,n}$, the raw processing capacity of each core $c_{n}$, 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 queuing delay associated with each link. Constraint (A8) helps to set the variable $u_{v,e}$, which indicates if there is any traffic that is routed through link $e$. Specifically, constraint (A9) imposes a deadline $\theta_e$ on each service’s end-to-end delay, which is calculated by summing up the queuing, processing and propagation delays. Constraint (A10) enforces that the number of deployed instances for a
Formulation 1 Co-located SFC Orchestration

\[
\text{Minimize } \sum_{n} \mathcal{P}_n \quad \text{(A1)}
\]

\[
\mathcal{P}_{n} = \sum_{\mathcal{S} \in \mathcal{U}} \sum_{f \in \mathcal{F}} \sum_{n \in \mathcal{N}_{\text{server}}} \mathcal{B} \cdot \mathcal{A} \quad \forall f \in F, \forall n \in \mathcal{N}_{\text{server}} \quad \text{(A2)}
\]

\[
\mathcal{C}_{f,n} = \mathcal{C}_{f,n} + \mathcal{C}_{n} \quad \forall f \in F, \forall n \in \mathcal{N}_{\text{server}} \quad \text{(A3)}
\]

\[
\mathcal{c}_{f,n} \geq c_{f,n} - 1 \quad \forall f \in F, \forall n \in \mathcal{N}_{\text{server}} \quad \text{(A4)}
\]

\[
\sum_{e \in \mathcal{E}} m_{f,n} \leq M_n \quad \forall n \in \mathcal{N}_{\text{server}} \quad \text{(A5)}
\]

\[
\mathcal{b}_{f,n} = \mathcal{w}_{f,n} \quad \forall e \in E \quad \text{(A6)}
\]

\[
\mathcal{b}_{f,n} = \mathcal{b}_{f,n} - \mathcal{b}_{f,n} \quad \forall e \in E \quad \text{(A7)}
\]

\[
\sum_{e \in \mathcal{E}} \sum_{n \in \mathcal{N}_{\text{server}}} \mathcal{a}_{n} \cdot \mathcal{d}_{f,n} \leq \theta_{f} \quad \forall f \in U \quad \text{(A8)}
\]

\[
\sum_{n \in \mathcal{N}_{\text{server}}} \mathcal{a}_{n} \leq \mathcal{K}_{f} \quad \forall f \in F \quad \text{(A9)}
\]

\[
\mathcal{a}_{n} = 1, \forall S \in U, \forall n \in \mathcal{N}_{\text{server}} \quad \text{(A10)}
\]

\[
\mathcal{a}_{n} = 0, \forall S \in U, \forall n \in S \quad \text{(A11)}
\]

\[
\sum_{e \in \mathcal{E}} \mathcal{l}_{f,n} + \mathcal{d}_{f,n} \leq \mathcal{c}_{f,n} \quad \forall e \in E \quad \text{(A12)}
\]

\[
\sum_{e \in \mathcal{E}} \mathcal{l}_{f,n} + \mathcal{d}_{f,n} \leq \mathcal{c}_{f,n} \quad \forall e \in E \quad \text{(A13)}
\]

\[
\mathcal{a}_{n} \geq \mathcal{i}_{f,n}, \forall n \in \mathcal{N}_{\text{server}}, \forall f \in F \quad \text{(A14)}
\]

\[
\mathcal{i}_{f,n} \geq 0, \forall f \in F, \forall n \in \mathcal{N}_{\text{server}} \quad \text{(A15)}
\]

\[
\mathcal{P}_{n} = \sum_{\mathcal{S} \in \mathcal{U}} \sum_{f \in \mathcal{F}} \sum_{n \in \mathcal{N}_{\text{server}}} \mathcal{B} \cdot \mathcal{A} \quad \forall f \in F, \forall n \in \mathcal{N}_{\text{server}} \quad \text{(A16)}
\]

\[
\mathcal{P}_{n} = \sum_{\mathcal{S} \in \mathcal{U}} \sum_{f \in \mathcal{F}} \sum_{n \in \mathcal{N}_{\text{server}}} \mathcal{B} \cdot \mathcal{A} \quad \forall f \in F, \forall n \in \mathcal{N}_{\text{server}} \quad \text{(A17)}
\]

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 \(a_{e,n}\) 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

\[
\text{Minimize } \sum_{n} \mathcal{P}_n m_n \quad \text{(B1)}
\]

\[
\sum_{n \in \mathcal{N}} m_{e,n} = 1, \forall S \in U, \forall e \in S_{n} \quad \text{(B2)}
\]

\[
m_{e} = \sum_{\mathcal{S} \in \mathcal{U}, e \in \mathcal{S}_{n}} \left[ \mathcal{b}_{f,n} \cdot \mathcal{a}_{n,e} / \sum_{n \in \mathcal{N}} \mathcal{C}_{n} \right] \quad \forall n \in \mathcal{N} \quad \text{(B3)}
\]

\[
m_{e} \leq M_{n} \quad \forall n \in \mathcal{N} \quad \text{(B4)}
\]

\[
r_{e}^\text{bw} = \sum_{\mathcal{S} \in \mathcal{U}, e \in \mathcal{S}_{n}} l_{j,e} \quad \forall e \in E \quad \text{(B5)}
\]

\[
r_{e}^\text{bw} \leq c_{e,\text{bw}} \quad \forall e \in E \quad \text{(B6)}
\]

\[
\sum_{e \in \mathcal{E}, e \in \mathcal{E}} l_{j,e} - \sum_{e \in \mathcal{E}, e \in \mathcal{E}} l_{j,e} = a_{j,\text{dest},n} - a_{j,\text{source},n} \quad \forall n \in N, \forall S \in U, \forall e \in E \quad \text{(B7)}
\]

\[
\sum_{j \in \mathcal{J}, e \in \mathcal{E}} l_{j,e} \leq \theta_{f} \quad \forall S \in U \quad \text{(B8)}
\]

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., \(j_{\text{dest}} = \text{egress} \)) 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 \(\theta_{f}\) 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 \(r_{e}^{\text{bw}}\) is assumed to lie within an interval \([r_{e}^{\text{bw}} - \delta_{r_{e}^{\text{bw}}}, r_{e}^{\text{bw}} + \delta_{r_{e}^{\text{bw}}}]\), where \(\delta_{r_{e}^{\text{bw}}}\) and \(\delta_{r_{e}^{\text{bw}}}\) denote the nominal value of \(r_{e}^{\text{bw}}\) and its maximum deviation, respectively. We assume that in the worst-case at most \(\Gamma\) demands can simultaneously deviate maximally from their nominal values, i.e.,

\[
\sum_{j} \left| r_{e}^{\text{bw}} - r_{e}^{\text{bw}} \right| \leq \Gamma. \quad \text{(R1)}
\]

\(\Gamma\) 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 \(\Gamma\) results in a less proactive solution that may lead to frequent resource reprovisioning at runtime. By setting a large value for \(\Gamma\), relative to the number of traffic demands, runtime reprovisioning can be completely avoided at the expense of a more costly solution. In this sense, \(\Gamma\) 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 $\Gamma$ demands does not make the model infeasible, it is necessary and sufficient that every resource constraint is protected against maximal deviations of any $\Gamma$ demands associated with that constraint.

**Proof.** It is trivial that when each constraint is individually protected from the worst-case deviation of $\Gamma$ demands associated with it, then regardless of which $\Gamma$ 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 $\Gamma$ demands to ensure the feasibility of the model. Assume by contradiction that we do not need to consider the worst-case deviation of $\Gamma$ demands at each constraint. Then there exists a set of $\Gamma$ demands, which provides the worst-case demand deviation at a resource constraint but is not considered. If the $\Gamma$ 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. $\blacksquare$

**Robust Optimization.** Theorem 1 implies that to account for the deviations of any $\Gamma$ demands, we must consider the maximal $\Gamma$ demand deviations associated with each capacity constraint. The problem is then reduced to an instance of $\Gamma$-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:

$$m_n = \sum_{\tilde{S} \in U} \sum_{v \in \tilde{S} \cup v_{\text{progress}}} m_{v,n} a_{v,n}, \forall n \in N,$$  

(R7)

with its linear robust counterpart being:

$$m_n = \gamma_n + \sum_{\tilde{S} \in U} \sum_{v \in \tilde{S} \cup v_{\text{progress}}} \tilde{m}_{v,n} a_{v,n} + \delta_{v,n}, \forall n \in N,$$  

(R8)

where $\gamma_n$, $\delta_{v,n}$, and $\tilde{m}_{v,n}$ 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 $P_{r_{i}^{\text{vio}}}$ denote the probability that the resource constraint $i$ is violated. Assume that the traffic demands are symmetrically distributed over their corresponding ranges, we have:

$$P_{r_{i}^{\text{vio}}} \leq \begin{cases} 0 \quad & \text{if } \Gamma \geq M_i, \frac{\Gamma^2}{2M_i} + \rho_i \quad & \text{if } 1 \leq \Gamma < M_i. \end{cases} \quad \text{(R10)}$$

where,

$$M_i = \left\lfloor \frac{\text{capacity of resource } i}{\text{minimum nominal demand}} \right\rfloor \leq \sum_{\tilde{S} \in U} |S_J|.$$  

(R11)

**Proof.** In the first case, when $\Gamma \geq M_i$, 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 $P_{r_{i}^{\text{vio}}} = 0$. For $1 \leq \Gamma < M_i$, 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]:

$$P_{r_{i}^{\text{vio}}} \leq \exp \left( \sum_{j \in \{\tilde{S} \in U \mid S_J\}} r_{j,\text{dest}} - \rho_i \right),$$  

(R12)

where,

$$\kappa_{j,i} = \begin{cases} \frac{r_{j,\text{dest}}}{r_{i,\text{dest}}} \quad & \text{if } j \in Q_i^*, \\ \frac{\tilde{r}_{j,\text{dest}}}{r_{i,\text{dest}}} \quad & \text{if } j \in C_i \setminus Q_i^*, \end{cases}$$  

(R13)

$$\tau^* = \arg\min_{\gamma \in Q_i^*} \{\tau_{r_{i}^{\text{vio}}} \}. \quad \text{(R14)}$$

We have $\rho$ is a constant, $C_i$ is the set of uncertain coefficients associated with resource constraint $i$ while $Q_i^*$ is the subset of coefficients that are selected to take their maximum deviations, and $\tau^*$ 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 $i_{j,e}$, 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., \( \forall i, C_i = \{ \bigcup_{S \in U} S_j \} \).

Assume \( 1 \leq \Gamma < M_i \), we have \( |Q^*_i| = \min\{\Gamma, M_i\} = \Gamma \). 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 \( j \in C_i \setminus Q^*_i \) 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,

\[
\sum_{j \in C_i \setminus Q^*_i} \frac{\hat{r}_{j,i}}{\hat{r}_{j,i}^*} \leq M_i - \Gamma \quad (R15)
\]
as \( \hat{r}_{j,i}/\hat{r}_{j,i}^* \leq 1 \) by the definition of \( \tau^* \) in (R14), we obtain that

\[
\sum_{j \in C_i} \nu_{j,i} \leq M_i. \quad (R16)
\]

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

\[
P_{\text{v},i,n}^* \leq \exp\left( \frac{M_i \rho^2}{2} - \rho \right) = \exp\left( -1/2M_\Gamma \right), \quad (R17)
\]
in which the parameter constant \( \rho \) is set to \( \Gamma_i/M_i \).

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 MISOCOP

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 MISOCOP, which can be solved using standard optimization solvers such as Gurobi [24].

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

\[
\begin{align*}
\hat{d}^\text{bwd} & \geq w_{e}, \forall e \in E, \quad (A19) \\
h^\text{bwd} & = c^\text{bwd} - \hat{d}^\text{bwd}, \forall e \in E. \quad (A20)
\end{align*}
\]

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

\[
2w^2_e + (\hat{d}^\text{bwd})^2 + (h^\text{bwd})^2 \leq (\hat{d}^\text{bwd} + h^\text{bwd})^2, \forall e \in E. \quad (A21)
\]

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:

\[
\begin{align*}
2\hat{d}^\text{fwd}_{j,n} & + \hat{d}^\text{proc}_{j,n}^2 + (h^\text{proc}_{j,n})^2 \leq (\hat{d}^\text{proc}_{j,n} + h^\text{proc}_{j,n})^2, \forall f \in F, \forall n \in N_{\text{server}}, \quad (A22) \\
h^\text{proc}_{j,n} & = c^\text{proc}_{j,n} - \hat{d}^\text{proc}_{j,n}, \forall f \in F, \forall n \in N_{\text{server}}. \quad (A23)
\end{align*}
\]

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

\[
\begin{align*}
\sum_{j \in S_j \in E} l_{j,n}(d^\text{bwd}_{e} + d^\text{prp}_{e}) + \\
\sum_{v \in S \in N_{\text{server}}} \sum_{n \in N_{\text{server}}} a_{\text{v},n} \hat{d}^\text{proc}_{j,n} \leq \theta_S, \forall S \in U. \quad (A24)
\end{align*}
\]

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

\[
\begin{align*}
\hat{d}^\text{bwd}_{j,n} & \leq (1 - l_{j,n})M_1, \forall n \in N_{\text{server}}, \forall v \in V, \forall j \in S_j, \quad (A25) \\
l_{j,n} \hat{d}^\text{bwd}_{j,n} & \leq l_{j,n}M_1, \forall e \in E, \forall n \in N_{\text{server}}, \forall j \in S_j. \quad (A26) \\
l_{j,n} \hat{d}^\text{bwd}_{j,n} & \leq d^\text{bwd}_{e}, \forall e \in E, \forall n \in N_{\text{server}}, \forall j \in S_j. \quad (A27)
\end{align*}
\]

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

\[
\begin{align*}
\hat{d}^\text{proc}_{v,n} & \leq (1 - a_{\text{v},n})M_2, \forall n \in N_{\text{server}}, \forall v \in V, \forall j \in S_j, \quad (A28) \\
a_{\text{v},n} \hat{d}^\text{proc}_{v,n} & \leq a_{\text{v},n}M_2, \forall n \in N_{\text{server}}, \forall j \in S_j, \quad (A29) \\
a_{\text{v},n} \hat{d}^\text{proc}_{v,n} & \leq d^\text{fwd}_{e}, \forall e \in E, \forall n \in N_{\text{server}}, \forall j \in S_j. \quad (A30)
\end{align*}
\]

For this to work, \( M_1 \) and \( M_2 \) should be set to sufficiently large values. Although we rely on the program to compute \( d^\text{proc}_{v,n} \) and \( d^\text{bwd}_{j,n} \), we know that these individual delay components should not be greater than the deadline \( \theta_S \) 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 \( d^\text{proc}_{v,n} \) and \( d^\text{bwd}_{j,n} \) in a feasible solution. The end-to-end delay requirement now can be expressed as the following linear constraint:

\[
\sum_{j \in S_j \in E} \sum_{e \in E} \hat{d}^\text{bwd}_{j,n} + l_{j,n}d^\text{prp}_{e} + \sum_{v \in S} \sum_{n \in N_{\text{server}}} \sum_{f \in F} \sum_{t(v)=f} d^\text{proc}_{v,n} \leq \theta_S, \forall S \in U. \quad (A31)
\]

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:

\[
\sum_{j \in S_j \in E} \sum_{e \in E} \hat{d}^\text{bwd}_{j,n} + l_{j,n}d^\text{prp}_{e} + \sum_{v \in S} \sum_{n \in N_{\text{server}}} \sum_{f \in F} \sum_{t(v)=f} d^\text{proc}_{v,n} - \theta_S \leq z_{S}, \forall S \in U. \quad (A32)
\]

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

\[
\text{Minimize } \sum_{n \in N} P_n + \lambda \sum_{S \in U} z_{S}. \quad (A33)
\]

where \( \lambda \) is a parameter that models the importance of fulfilling the hard deadlines. Specifically, by setting \( \lambda \) to a
sufficiently large value, the model is forced to meet the hard deadlines \( (i.e., z_s \to 0 \text{ as } \lambda \to \infty) \). 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 \( \lambda \) 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 MISOPC is given in Algorithm 1.

**Algorithm 1 SODL** (Exact Co-located SFC Orchestration)

\[
\begin{align*}
\text{Minimize} & \quad \sum_n P_n + \lambda \sum_S z_S \\
\text{subject to} & \quad (A3)-(A5), (R3) \text{ and } (R4) \text{ energy cstrs: } (A17) \\
& \quad (A10)-(A13) \text{ placement cstrs: } (A10)-(A13) \\
& \quad \text{routing cstrs: } (A15), (A16), (R5) \text{ and } (R6) \\
& \quad \text{delay cstrs: } (A19)-(A32).
\end{align*}
\]

**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 MISOPC \((i.e., a_{v,n}, f_{j,n}, w_{v,n}, c_{a,v,n}, m_{f,n})\) transforming it into a SOPC, 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.

**Fractional Placement.** The mapping of virtual nodes to the substrate nodes is given by the variable \( a_{v,n} \). We define the list \( V \) 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 \( v \in V \) and \( n \in N_v \), the algorithm selects the top \( \Upsilon \) 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 \( V \) (line 22), preventing them from being re-picked in future iterations. Once all virtual nodes have been mapped \((i.e., V = \emptyset)\), the main loop of the algorithm (lines 10 - 32) terminates. The parameter \( \Upsilon \) 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 \( \Upsilon \), 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 \( \Upsilon \) 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 \((i.e., v \notin V)\) 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 \( \Gamma \) 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.
of all VNFs and verifies that it is less than $M_n - |F_n| + 1$, where $M_n$ is the total number of available CPU cores and $|F_n|$ 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 $J$. In each iteration, only the virtual links that are still in $J$ are considered. RestrictRouting iterates through every possible mapping of each virtual link in $J$ 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 $|F_n|$ on the server.

**Theorem 3.** Preserving $|F_n| - 1$ 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 $\sum_{f \in F_n} m_{f,n} \leq M_n - (|F_n| - 1)$. Since certainly $|m_{f,n}| \leq m_{f,n} + 1$, it follows that $m_{f,n} + \sum_{f \in F_n \setminus \{f\}} |m_{f,n}| \leq M_n - (|F_n| - 1) + (|F_n| - 1)$, and thus $\sum_{f \in F_n} m_{f,n} \leq M_n$. □

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 $xM_{place}$. 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 set 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 $c_{v,n}$. 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

<table>
<thead>
<tr>
<th>Notation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$M_{\text{place}}$</td>
<td>Relaxed VNF placement model</td>
</tr>
<tr>
<td>$M_{\text{route}}$</td>
<td>Relaxed VNF routing model</td>
</tr>
<tr>
<td>$S_{\text{sorted}}$</td>
<td>Sorted DCs in descending order of their $c_n/P_n$</td>
</tr>
<tr>
<td>$S_{\text{path}}$, $S_{\text{mig}}$</td>
<td>Set of paths between DC $n_1$ and $n_2$</td>
</tr>
<tr>
<td>$P_{\text{route}}$, $P_{\text{mig}}$</td>
<td>Set of paths for routing flow $j$</td>
</tr>
<tr>
<td>$P_{\text{route}}$, $P_{\text{mig}}$</td>
<td>Set of paths that use substrate link $e$</td>
</tr>
<tr>
<td>$V_{\text{route}}$, $V_{\text{mig}}$</td>
<td>Set of virtual nodes (VNF requests) that are mapped to $n$</td>
</tr>
<tr>
<td>$E_{\text{route}}$, $E_{\text{mig}}$</td>
<td>Set of links whose capacities cannot be met in the irredicible system</td>
</tr>
</tbody>
</table>

Algorithm 3 xGSODL

Input: $M_{\text{place}}$, $\Gamma$, $\kappa$
Return: $(\mathcal{P}_n|n \in N)$, $(J_{\text{route}}|J \in E)$

1. $S_{\text{sorted}} \leftarrow \{n| \forall n \in N, c_n = n_{k+1}\}$
   \hspace{1cm} \Comment{Sort the DCs in descending order}
2. $S_{\text{path}}$, $S_{\text{mig}} \leftarrow \{n|\forall n \in N, c_n = n_{k+1}\}$
   \hspace{1cm} \Comment{Compute the k shortest paths}
3. $\mathcal{P}_n \leftarrow \{n|\forall n \in N, c_n, \forall n \in N\}$
4. while $\text{True}$ do
5. \hspace{1cm} $(\mathcal{P}_n|n \in N) \leftarrow \text{OptimalRouting}(\mathcal{P}_{\text{route}}, \Gamma)$
   \hspace{1cm} \Comment{Find initial placement}
6. \hspace{1.5cm} for $n \in S_{\text{sorted}} \leftarrow \text{False}$ do
7. \hspace{2.5cm} \Comment{Check the DC capacity and perform migration if necessary}
8. \hspace{3.5cm} if $\sum_{j \in J_{\text{route}} \in P_{\text{route}} \subseteq S_{\text{path}}} \leq \sum_{j \in J_{\text{route}} \in P_{\text{route}} \subseteq S_{\text{mig}}} m_n > M_{\text{mig}}$ then
9. \hspace{4.5cm} $\mathcal{P}_{\text{mig}} \leftarrow \{\text{Solve (B9) with } c_n = c_n, \forall n \in N\}$
10. \hspace{3.5cm} $\text{mig}, \mathcal{P}_{\text{mig}}$ \Comment{Perform VNF migration}
11. \hspace{2.5cm} end if
12. \hspace{1cm} end for
13. $\text{end if}$
14. $P_{\text{route}} \leftarrow \{p|\forall P_{\text{route}} \in N, \forall n \in S_{\text{path}} \subseteq S_{\text{route}} \subseteq S_{\text{mig}}\}$
15. $\text{end for}$
16. $\text{end if}$
17. $(f \in \mathcal{E}, \forall e \in \mathcal{E}, \forall J \in P_{\text{route}} \subseteq S_{\text{route}} \subseteq S_{\text{mig}})$
18. \hspace{1cm} $\text{Compute the irredicible system}$
19. $\text{end for}$
20. $(N_e|\forall e \in \mathcal{E}, \forall \mathcal{E} \subseteq S_{\text{route}} \subseteq S_{\text{mig}})$
21. $\text{end for}$
22. $\text{end for}$
23. $\text{end for}$
24. $J_{\text{route}} = \{J|\forall J \in P_{\text{route}} \subseteq S_{\text{route}} \subseteq S_{\text{mig}}\}$
25. $\text{end for}$
26. $\text{end if}$
27. $\text{end if}$
28. $\text{end if}$
29. $\text{end if}$
30. $\text{end if}$
31. $\text{end for}$
32. $\text{end if}$
33. $\text{end if}$
34. $\text{end if}$
35. $\text{end if}$
36. $\text{end if}$
37. $\text{end for}$
38. $\text{end for}$
39. $\text{end for}$
40. $\text{end for}$
41. $\text{end if}$
42. $\text{end while}$
43. $\text{return}$ $(\mathcal{P}_n|n \in N)$, $(J_{\text{route}}|J \in E)$

Optimization Analysis. Let $P_{\text{opt}}$ and $xGSO_{\text{opt}}$ 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 $\sum_{j \in J_{\text{route}}} \sum_{j \in J_{\text{route}}}$ to indicate the summation over all virtual links (excluding those destined to an egress) and $\sum_{j \in J_{\text{route}}}$ to indicate the summation over the virtual links whose demands are among the $\Gamma$ most deviated demands.

Theorem 4. $P_{\text{opt}} - P_{\text{nominal}} \leq \frac{1}{2} + \frac{1}{\alpha(\Gamma)} \frac{P_{\text{opt}}}{P_{\text{nominal}}} w_c$, where $b = \text{argmax}_{\mathcal{E}} \{c_n/P_n | \forall n \in N\}$, $w = \text{argmin}_{\mathcal{E}} \{c_n/P_n | \forall n \in N\}$, and $\alpha(\Gamma) = \frac{\sum_{j \in J_{\text{route}}} \sum_{j \in J_{\text{route}}}}{\sum_{j \in J_{\text{route}}} \sum_{j \in J_{\text{route}}}} = \frac{1}{\sum_{j \in J_{\text{route}}} \sum_{j \in J_{\text{route}}}} + 1 + \frac{1}{\sum_{j \in J_{\text{route}}} \sum_{j \in J_{\text{route}}}}$.
the solution. Specifically, let packing

\[ \left( \frac{1}{\sum_j \frac{r_j}{c_j \sigma_j}} \right)^2 \leq \frac{p_{\text{route}}}{p_{\text{route}}^*} \leq \frac{p_{\text{route}}^*}{p_{\text{route}}} \]  

be achieved by \( n_j \), respectively, then

\[ \frac{1}{\sum_j \frac{r_j}{c_j \sigma_j}} \leq \frac{p_{\text{route}}}{p_{\text{route}}^*} \leq \frac{p_{\text{route}}^*}{p_{\text{route}}} \]  

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.

**Theorem 5.** If \( \exists S \in \mathcal{U}, j \in S, p \in \mathcal{P}_j, x_j^P \notin \{0, 1\} \) where \( E' = \{e \in \mathcal{P}_j \mid \forall S \in \mathcal{U}, \forall j \in S, \forall e \in E \} \).

**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., \( \forall S \in \mathcal{U}, j \in S, p \in \mathcal{P}_j, x_j^P \notin \{0, 1\} \). We have \( \exists S \in \mathcal{U}, j \in S, p \in \mathcal{P}_j, x_j^P = \{0, 1\} \). 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 \( j.\text{source} \) and \( j.\text{dest} \) 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.

**Theorem 5.** If \( \exists S \in \mathcal{U}, j \in S, p \in \mathcal{P}_j, x_j^P \notin \{0, 1\} \) where \( E' = \{e \in \mathcal{P}_j \mid \forall S \in \mathcal{U}, \forall j \in S, \forall e \in E \} \).

**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., \( \forall S \in \mathcal{U}, j \in S, p \in \mathcal{P}_j, x_j^P \notin \{0, 1\} \). We have \( \exists S \in \mathcal{U}, j \in S, p \in \mathcal{P}_j, x_j^P = \{0, 1\} \). 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 \( b \).

### Formulation 3 \( M_{\text{route}} \)

\[ \text{Minimize} \sum_{S \in \mathcal{U}} \sum_{j \in S} \sum_{p \in \mathcal{P}_j} x_j^P \theta_j^{\text{dep}} \]  

\[ \sum_{p \in \mathcal{P}_j} x_j^P = 1, \forall S \in \mathcal{U}, \forall j \in S, \]  

\[ \sum_{j \in S, p \in \mathcal{P}_j} x_j^P < c_e, \forall e \in \mathcal{E}, \]  

\[ \sum_{j \in S, p \in \mathcal{P}_j} x_j^P \theta_j^{\text{dep}} \leq \theta_S, \forall S \in \mathcal{U}, \]  

\[ x_j^P \in \{0, 1\}, \forall S \in \mathcal{U}, \forall j \in S, \forall p \in \mathcal{P}_j \]

In practice, \( |E'| \) is typically smaller than \( \sum_{S \in \mathcal{U}} |S| - 1 \) 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 flow \( j \) (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 \texttt{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, the function `FlowMigrate` iteratively checks through the set of paths in \( P_j \) 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 \( j.\text{source} \) or \( j.\text{dest} \) 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 \( P_j \) 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 \( j.\text{source} \) or \( j.\text{dest} \) updated to omit the current location. For all scenarios above, the decisions on whether to select \( j.\text{source} \) or \( j.\text{dest} \) 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 are strictly enforced by SODL as \( \lambda = 10^6 \) 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.

![Figure 3: Exact model evaluation settings.](image)

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

<table>
<thead>
<tr>
<th>Server configurations (4 classes)</th>
<th>No. of cores (( M_n ))</th>
<th>4 cores</th>
<th>6 cores</th>
<th>8 cores</th>
<th>10 cores</th>
</tr>
</thead>
<tbody>
<tr>
<td>Raw core capacity (( P_{\text{act}}^{(c)} ))</td>
<td>1.0 Gbps</td>
<td>1.0 Gbps</td>
<td>1.0 Gbps</td>
<td>1.0 Gbps</td>
<td></td>
</tr>
<tr>
<td>Idle power (( P_{\text{idle}}^{(c)} ))</td>
<td>70 Watts</td>
<td>75 Watts</td>
<td>80 Watts</td>
<td>100 Watts</td>
<td></td>
</tr>
<tr>
<td>Max power (( P_{\text{max}}^{(c)} ))</td>
<td>220 Watts</td>
<td>225 Watts</td>
<td>300 Watts</td>
<td>350 Watts</td>
<td></td>
</tr>
<tr>
<td>Switch configurations (1 class)</td>
<td>120.0 Gbps</td>
<td>120.0 Gbps</td>
<td>120.0 Gbps</td>
<td>120.0 Gbps</td>
<td></td>
</tr>
<tr>
<td>Max power (( P_{\text{max}}^{(s)} ))</td>
<td>60 Watts</td>
<td>60 Watts</td>
<td>60 Watts</td>
<td>60 Watts</td>
<td></td>
</tr>
<tr>
<td>VNF configurations (5 types)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VNF</td>
<td>Abbreviation</td>
<td>Processing coeff. (( \lambda_j ))</td>
<td>No. of licenses (( K_j ))</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Traffic Monitor</td>
<td>TM</td>
<td>1.0</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fire Wall</td>
<td>FW</td>
<td>0.9</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Intrusion Detection System</td>
<td>IDS</td>
<td>0.7</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>WAN Optimizer</td>
<td>WO</td>
<td>0.6</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Video Optimizer</td>
<td>VO</td>
<td>0.6</td>
<td>3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Effect of Demand Fluctuations.** The goal of this experiment 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 \( \Gamma \), which is the parameter that controls the proactivity of the algorithm. We consider three demand profiles with relative maximum deviations of \( \tilde{\gamma}_j/\bar{\gamma}_j = 10\%, 30\%, \) 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.

![Figure 4: Performance of SODL under different proactiveness level.](image)
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 $\Gamma$ 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 $\Gamma$ for the three demand profiles. As evident from the figure, the energy consumption increases monotonically with $\Gamma$. This is expected since by increasing $\Gamma$, the model becomes more proactive and provisions more network resources to cope with higher future demand fluctuations, thus increasing the energy consumption.

Comparison 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 $\rho$ to 1 results in a fully proactive orchestration, while setting $\rho$ 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 $\rho$. Figs. 5(a) and 5(b), respectively, compare the robustness and energy cost of the baseline using three values for $\rho$ (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 $\Upsilon$ 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 $\Upsilon$. Recall that $\Upsilon$ 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 $\Upsilon$, the runtime reduces almost by half. Therefore, depending on the problem scale, $\Upsilon$ can be set to allow the algorithm to run in a reasonable time. Though this benefit comes at a price. As $\Upsilon$ 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.
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 $\lambda$ is set to be sufficiently large. When $\lambda$ 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 $\lambda$ 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 $\Gamma$ 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 $\theta_S$. The percentiles are computed over 200 runs of xGSODL on randomly generated demand profiles. When $\theta_S \in \{500, 600\}$, 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 $\theta_S$ is set to be more stringent (i.e., $\theta_S \in \{450, 550\}$). As there is less flexibility in terms of delay, it is more likely that a computed
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 Integrity. This experiment studies the effects of parameters such as the degree of support for each virtual node \(N_v\), the total traffic demand \(\sum_{S \in \mathcal{S}} |S_j|\), and the number of precomputed paths \(K\) on the integrality of the relaxed \(M_{\text{route}}\). For each setting of these parameters, we make 200 runs of xGSODL on randomly generated demand vectors and record the following information: \(\eta_1\) - the percentage of runs in which the relaxed solutions are all integral, \(\eta_2\) - the average percentage of variables that are not integral, and \(\eta_3\) - the average number of non-integral solutions. All collected data are reported in Table IV. As \(|N_v|\) becomes bigger, we observe that \(\eta_1\) decreases for all considered traffic loads and numbers of precomputed paths. When \(|N_v|\) increases, constraint (B17) in \(M_{\text{route}}\) 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 \(M_{\text{route}}\), 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 \(\eta_2\) and \(\eta_3\) indicate a vast majority of the relaxed variables are still integers. We further observe that \(\eta_1\) 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., queuing delay).

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


Copyright (c) 2020 IEEE. Personal use is permitted. For any other purposes, permission must be obtained from the IEEE by emailing pubs-permissions@ieee.org.