Proactive Service Orchestration with Deadline

Minh Nguyen, Mahdi Dolati and Majid Ghaderi
Department of Computer Science, University of Calgary

***Abstract*—In network function virtualization, network services are implemented as service function chains (SFCs). An extensive body of work exists on SFC orchestration, although a vast majority of them consider *reactive* algorithms that reprovision resources in response to service demand fluctuations. As such, they result in unpredictable and often significant delays that negatively affect the performance of delay sensitive SFCs. In this paper, we consider *proactive* SFC orchestration and develop exact and approximate algorithms that perform well under general service demands without requiring frequent resource reprovisioning. Specifically, we first formulate SFC orchestration with deadline as a mixed integer non-linear program and show that it can be reduced to a second-order cone program, which can be solved using standard optimization software, albeit for small problem instances. We then design an approximate algorithm for large problem instances by applying iterative rounding and variable fixing techniques to the exact problem formulation. 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 provisioning-reprovisioning trade-off. We also compare the performance of our algorithms against two recently proposed algorithms called FFCA [1] and MaxZ [2].**

I. Introduction

A. Background and Motivation

Network function virtualization (NFV) [3] is an emerging paradigm, which enables rapid and scalable deployment of network services. In this paradigm, network functions, traditionally performed by specialized hardware appliances, are softwarized and deployed on an NFV infrastructure (NFVI). Typically, these software-based network functions, commonly referred to as virtual network functions (VNFs), are installed and run on virtual machines (VMs) hosted in an enterprise cloud.

To realize the benefits of NFV, several technical challenges need to be addressed. One important challenge is the orchestration of service function chains (SFCs) [4]. 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 operational cost of orchestrating the SFCs in the NFVI by provisioning just enough resources (*e.g.*, physical servers) to satisfy service demands according to prespecified service level agreements (SLAs).

Optimal provisioning of cloud resources for SFC orchestration, 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.*, [5]) 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.

In this paper, we consider proactive SFC orchestration and develop exact and approximate orchestration algorithms that perform well under general service demands without requiring frequent resource reprovisioning.

B. Related Work

Prior works that are most relevant to ours are reviewed below, where proactive SFC orchestration algorithms are divided into *online* and *robust* algorithms.

Reactive algorithms. Extensive surveys on reactive SFC orchestration algorithms are presented in [6] and [7]. The common theme in these algorithms is the deterministic modeling of service demands in an offline setting during the service provisioning. Typically, the nominal or worst-case service demands are assumed to devise an initial orchestration model. Later, at run-time, VNFs are dynamically reprovisioned (*e.g.*, scaled up and down) in response to demand fluctuations to meet the prespecified service requirements. As discussed earlier, while this approach is suitable for delay-tolerant services, it results in poor performance for delay-sensitive services.

Online algorithms. The key feature of online algorithms is the assumption that no information about future service demands is available, yet the orchestration should be conducted such that any possible future service demands can be satisfied without reprovisioning. Since no information about future demands is assumed, the worst-case performance of online algorithms (such as those presented in [8] and [9]) could be particularly bad. On the other hand, when reasonably accurate predictions about future demands could be obtained, techniques from online convex optimization can be used to design more efficient online algorithms [10–12]. However, if the predictions are not accurate, these algorithms result in inefficient orchestrations.

Robust algorithms. While it is not realistic to assume that full knowledge of future service demands is available a priori, often in practice partial knowledge about demands could be obtained as part of SLAs, application requirements, admission control policies, or past history. Such information, however noisy or incomplete, if incorporated in SFC orchestration, results in more efficient provisioning decisions. To this end, a few works have studied SFC orchestration with demand uncertainty (*e.g.*, [1], [13–15]). Specifically, the work in [13] formulates SFC orchestration as a robust optimization problem with the objective

of maximizing the NFVI operator's profit. More relevant to our work is [14], in which the authors propose a robust orchestration model that aims at minimizing the energy consumption of the NFVI. This model, however, is computationally expensive and can only be solved for small problem instances. To speed up solving this model, a heuristic approach is presented in [15], in which the placement, routing and resource allocation sub-problems are solved separately. While the proposed heuristic can be applied to large problem instances, it results in sub-optimal solutions as it considers orchestration sub-problems separately. In a follow-up work, the authors consider SFC orchestration in 5G virtualized infrastructures, and model the problem as a robust binary optimization problem, which can be solved efficiently [1]. This work considers service delay but in a rather simplified model, where a deadline is considered for each pair of consecutive VNFs in a hop-by-hop manner and only the propagation delay is taken into account. In our work, we consider the end-to-end delay over the entire SFC, and consider not only the fixed propagation delays but also load-dependent queueing and processing delays. As we show in Section III, considering such a detailed delay model changes the structure of the SFC orchestration problem by introducing non-linear constraints into the formulation. This requires entirely different set of techniques to model and solve the SFC orchestration problem.

C. Our Work

We consider the SFC orchestration problem with the objective of minimizing the energy consumption of the NFVI. Specifically, given a list of SFCs representing the requested services and an NFVI consisting of servers and switches, our objective is to optimally deploy the VNFs required to provide these services such that the energy consumed by the NFVI is minimized while meeting a deadline for each service. To compute energy consumption, we consider a detailed model, which accounts for the energy consumed by servers and switches, and includes both load-dependent and constant energy consumption of the infrastructure elements. To account for deadlines, we consider an end-to-end delay model that considers load-dependent processing and queuing delays as well as constant propagation delays across the entire SFC chains.

To solve the SFC orchestration problem, we develop exact and approximate algorithms that perform well under general service demands without requiring frequent reprovisioning. We assume that some knowledge about future traffic demands, *i.e.*, the nominal traffic demand and maximum demand deviation, is available and formulate the problem as a robust optimization problem. A key feature of our algorithms is that they can achieve any desired trade-off between the proactivity of the computed orchestration and its energy cost. They can compute SFC orchestrations that minimize the NFVI energy cost but cause frequent reprovisioning at run-time. Alternatively, they can compute orchestrations that avoid run-time reprovisioning completely at the cost of increased NFVI energy cost.

Our main contributions in this paper are summarized below:

- We formulate the proactive SFC orchestration with deadline as a mixed-integer non-linear program. Then, we show that the problem can be transformed into a mixed-integer second-order cone program (MISOCP), which can be solved for small problem instances.
- To tackle the computational complexity of the MISOCP formulation, we apply several techniques including iterative rounding and variable fixing to the exact formulation, which results in an approximate model that can be efficiently solved for large problem instances.
- 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 also compare our approximate algorithm against two recently proposed algorithms, namely the First Fit Clustering Allocation (FFCA) [1] and MaxZ [2] algorithms.

D. Paper Organization

In Section II, we describe different components of our model. The mathematical formulation is presented in Section III. Sections IV and V present the exact and approximate algorithms, respectively. The simulation results are shown in Section VI. Finally, Section VII concludes the paper. **Due to space limitation, some technical details and derivations are omitted from this version and can be found (online) in [16].**

II. System Model

In this section, we discuss our assumptions and describe the models for various components of the NFVI.

A. NFVI Model

The substrate network is modeled as a directed graph $G^{phy} = (N, E)$, where N and E, respectively, are the sets of physical nodes and links. A physical node can either be a switch or a server. We have $N = N_{server} \cup N_{switch}$, in which N_{server} and N_{switch} denote the sets of servers and switches, respectively. Also, the set of all VNFs is denoted by F.

Each server is capable of hosting any VNF as long as it has sufficient capacity. The processing capacity of server n is determined by the number of available CPU cores M_n and the processing capacity of each core c_n^{cpu}. Given the processing rate requirement $r_{f,n}^{proc}$ for VNF f at server n, we allocate $m_{f,n}$ CPU cores to the VM that runs the function f. The allocated CPU cores to the VM should provide enough processing capacity $c_{f,n}^{proc}$ to meet the processing rate requirement $r_{f,n}^{proc}$. Unlike previous works which assume that the processing rate requirement is fixed and solely dependent on the type of VNF (*e.g.*, [1], [8], [11], [13–15]), the processing rate requirement in our model is a function of the incoming traffic rate. We also assume that all links in the substrate network are full-duplex, and model each physical link as two equal-capacity directed links in opposite directions. We let r_e^{bwd} denote the aggregate rate of traffic on link e, which should not be greater than the link capacity c_e^{bwd}.

B. VNF Model

The NFVI operator holds a set of VNF images corresponding to the offered services. The VNF images are deployed on the servers in the NFVI to provide a set of services. We define the binary variable $i_{f,n}$ to indicate if server n hosts an instance of VNF f. 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. Each VNF f has a processing coefficient σ_f, ranging between 0 and 1, that indicates its processing complexity. Simple VNFs such as Firewall and NAT, which do not require much processing, have their σ_f close to one. On the other hand, computationally more demanding VNFs such as DPI and WAN optimizer have smaller values of σ_f.

C. SFC Model

The set of SFCs to be orchestrated in the NFVI is denoted by $\mathcal{U}$. Each SFC S is modelled as a virtual network represented by a directed graph $G^{virt} = (S_V, S_J)$, where S_V and S_J are, respectively, the sets of virtual nodes and links. The virtual nodes include the ingress/egress nodes as well as the required VNFs (see Fig. 3(a) for example SFCs). We use lower case letter j to denote an arbitrary virtual link and v to represent an arbitrary virtual node. While multiple instances of each VNF are deployed on different servers, we assume that, with a proper scheduling mechanism (*e.g.*, [17]), multiple SFCs can share a VNF instance deployed on a server.

The binary variable $a_{v,n}$ is used to indicate whether virtual node v is mapped to physical node n. Also, define the variable $l_{j,e}$ to denote if physical link e is responsible for routing the traffic of virtual link j. Each virtual link j has a traffic demand r_j^{bwd}. In this work, we assume single-path routing, so each virtual link is mapped to a set of physical links that together compose a path, connecting the physical nodes to where the two ends of the virtual link are mapped. For each virtual node v, we define the set of all substrate nodes where v can be mapped to as N_v. There are several uses of the set N_v. First, it gives a convenient way to restrict the placement of ingress and egress nodes to certain locations in the substrate network. Second, it allows the NFVI operator to narrow the possible locations of each VNF to a specific set of servers. This may be done due to security or administrative requirements associated with each VNF.

D. Demand Fluctuation Model

Each future service demand r_j^{bwd} is assumed to lie within an interval $[\bar{r}_j^{bwd} - \hat{r}_j^{bwd}, \bar{r}_j^{bwd} + \hat{r}_j^{bwd}]$, where $\bar{r}_j^{bwd}$ and $\hat{r}_j^{bwd}$ denote the nominal value of r_j^{bwd} and its maximum deviation, respectively. To control the level of proactivity of the model, we assume that at most Γ demands can simultaneously deviate maximally from their nominal values, *i.e.*,

$$\sum_j \left| \frac{r_j^{bwd} - \bar{r}_j^{bwd}}{\hat{r}_j^{bwd}} \right| \leq \Gamma. \tag{1}$$

A small Γ results in a less proactive solution that may lead to frequent reprovisioning at run-time. By setting a large value for Γ, relative to the number of traffic demands, run-time reprovisioning can be completely avoided at the cost of more energy consumption.

E. Energy Model

The energy consumption of both servers and switches is modelled by a linear function of their utilization [18]. While the switches are always active, we assume that the servers can be turned off if they do not host any VNF. The on/off status of server n is indicated by the binary variable o_n. Let P_n^{active} denote the power consumption of an active device in idle state. Then, the power consumption of device n, denoted by P_n, scales proportionally with its resource utilization. Let P_n^{max} denote the maximum power consumption of device n at full utilization.

F. Delay Model

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. The processing delay $d_{f,n}^{proc}$ of VNF f at 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 physical 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.

III. Problem Formulation

In this section, we formulate SFC orchestration as a constrained optimization problem. For clarity, we group the constraints into several groups as follows.

A. Resource Provisioning

In our model, the processing capacity required to deploy a VNF instance is not known a priori and can only be determined at the time of placement by summing the traffic demands of all service flows to be served by that VNF instance. Specifically, the minimum processing rate $r_{f,n}^{proc}$ required to deploy VNF f on server n is computed as follows:

$$r_{f,n}^{proc} = \sum_{S \in \mathcal{U}} \sum_{\substack{j \in S_J \\ t(j.dest)=f}} r_j^{bwd} \cdot a_{j.dest,n}, \ \forall f \in F, \forall n \in N_{server}, \tag{2}$$

where the function $t(j.dest)$ indicates the VNF that is required by virutal node $j.dest$. Only those service flows (*i.e.*, virtual links) that destined to VNF f and deployed on server n are considered in the calculation of $r_{f,n}^{proc}$. Given this processing requirement, the allocated processing capacity $c_{f,n}^{proc}$ is determined by the following constraints:

$$c_{f,n}^{proc} = \sigma_f \cdot m_{f,n} \cdot c_n^{cpu}, \ \forall f \in F, \forall n \in N_{server}, \tag{3}$$

$$r_{f,n}^{proc} \leq c_{f,n}^{proc}, \ \forall f \in F, \forall n \in N_{server}. \tag{4}$$

These constraints state that CPU cores allocated to run a VNF should provide enough processing capacity for it to at least meet its minimum processing requirement $r_{f,n}^{proc}$; though more CPU cores can be allocated to each VNF to speed up the processing, ensuring that the deadline for an SFC is met. However, the total number of allocated CPU cores must be less than the number of available CPU cores at the server:

$$\sum_{f \in F} m_{f,n} \leq M_n, \ \forall n \in N_{server}. \tag{5}$$

B. Delay Requirement

Let the binary variable w_e indicate if there is any traffic load on link e. The queueing delay d_e^{bwd} at link e is given by:

$$d_e^{bwd} = \frac{w_e}{c_e^{bwd} - r_e^{bwd}}, \ \forall e \in E, \tag{6}$$

$$w_e \geq l_{j,e}, \ \forall S \in \mathcal{U}, \forall j \in S_J, \forall e \in E. \tag{7}$$

When there is no traffic load, the queueing delay is 0. The processing delay $d^{proc}_{f,n}$ for VNF f on server n is given by:

$$d^{proc}_{f,n} = \frac{i_{f,n}}{c^{proc}_{f,n} - r^{proc}_{f,n}}, \ \forall n \in N_{server}. \tag{8}$$

The binary variable $i_{f,n}$ indicates whether VNF f is hosted on server n. For a VNF that is not hosted on a server, its processing delay on that server is 0. The end-to-end delay for each SFC S must satisfy its deadline denoted by θ_S:

$$\sum_{j \in S_J} \sum_{e \in E} l_{j,e}(d^{bwd}_e + d^{prop}_e) \ + \\ \sum_{v \in S_v} \sum_{n \in N_{server}} \sum_{\substack{f \in F \\ t(v)=f}} a_{v,n} d^{proc}_{f,n} \leq \theta_S, \ \forall S \in \mathcal{U}. \tag{9}$$

C. VNF Placement

The optimization model chooses the optimal number of instances of each VNF to deploy as well as where to deploy them. However, the total number of deployed instances for a VNF should be no more than the number of licenses available:

$$\sum_{n \in N_{server}} i_{f,n} \leq K_f, \ \forall f \in F, \tag{10}$$

where K_f denotes the number of licenses for VNF f. Except for the ingress and egress nodes, each virtual node in an SFC graph denotes a VNF required for that service. Each of these nodes must be mapped to a server, where the corresponding VNF is deployed, as enforced by the constraint:

$$a_{v,n} \ \leq i_{f,n}, \\ \forall S \in \mathcal{U}, \forall v \in S_V, \forall n \in N_{server}, \forall f \in F : t(v) = f. \tag{11}$$

Each virtual node v must be mapped to a substrate node in its set of possible locations N_v, as captured by:

$$\sum_{n \in N_v} a_{v,n} = 1, \ \forall S \in \mathcal{U}, \forall v \in S_V, \tag{12}$$

$$a_{v,n} = 0, \ \forall S \in \mathcal{U}, \forall v \in S_V, \forall n \in N : n \notin N_v. \tag{13}$$

D. Flow Routing

The problem of selecting paths for routing traffic between the deployed VNFs can be modelled as a *single-path multi-commodity flow* problem, which has a standard set of constraints. First, we need to ensure that the aggregate flow rate on each link does not exceed its capacity:

$$r^{bwd}_e = \sum_{S \in \mathcal{U}} \sum_{j \in S_J} r^{bwd}_j l_{j,e}, \ \forall e \in E, \tag{14}$$

$$r^{bwd}_e \leq c^{bwd}_e, \ \forall e \in E. \tag{15}$$

The aggregate flow on each link is computed as the sum of traffic rates of all service flows (*i.e.*, virtual links) that are routed through it as shown in (14). Second, we have a constraint to ensure the conservation of flows at each node:

$$\sum_{e : e.dest = n} l_{j,e} - \sum_{e : e.source = n} l_{j,e} = a_{j.dest,n} - a_{j.source,n}, \\ \forall n \in N, \forall S \in \mathcal{U}, \forall j \in S_J. \tag{16}$$

This constraint states that unless node n is either the source or destination of flow j, it only relays traffic of flow j.

E. Power Consumption

We consider the energy consumption for both servers and switches. To reduce the energy consumption, only those servers that have to host some VNFs are turned on. We use the following constraint to identify those servers that are active:

$$o_n \geq i_{f,n}, \quad \forall n \in N_{server}, \forall f \in F. \tag{17}$$

The energy consumption of active servers and switches is a function of their utilization. This relation is captured by the following linear model:

$$P_n = \begin{cases} o_n P^{active}_n + \dfrac{\sum_{f \in F} m_{f,n}}{M_n}(P^{max}_n - P^{active}_n) & \forall n \in N_{server} \\ P^{active}_n + \dfrac{\sum_{e.dest=n} r^{bwd}_e}{c^{bwd}_n}(P^{max}_n - P^{active}_n) & \forall n \in N_{switch}. \end{cases} \tag{18}$$

F. Demand Fluctuations

We apply the Γ-Robust optimization framework [19] to compute SFC orchestrations that remain feasible for all possible future service demands (those that conform with the demand specifications). To this end, we replace (2) and (14) in the original formulation by their linear robust counterparts (see our technical report [16] for details). Specifically, (2) is replaced by,

$$r^{proc}_{f,n} = \Gamma \alpha_{f,n} + \sum_{S \in \mathcal{U}} \sum_{\substack{j \in S_J \\ t(j.dest)=f}} \bar{r}^{bwd}_j a_{j.dest,n} + \beta_{j,f,n}, \\ \forall f \in F, \forall n \in N_{server}, \tag{19}$$

$$\alpha_{f,n} + \beta_{j,f,n} \geq \hat{r}^{bwd}_j a_{j.dest,n}, \\ \forall f \in F, \forall n \in N_{server}, \forall S \in \mathcal{U}, \forall j \in S_J : t(j.dest) = f, \tag{20}$$

while (14) is replaced by the following equations,

$$r^{bwd}_e = \Gamma \gamma_e + \sum_{S \in \mathcal{U}} \sum_{j \in S_J} l_{j,e} \bar{r}^{bwd}_j + \zeta_{j,e}, \ \forall e \in E, \tag{21}$$

$$\gamma_e + \zeta_{j,e} \geq \hat{r}^{bwd}_j l_{j,e}, \ \forall e \in E, \forall S \in \mathcal{U}, \forall j \in S_J. \tag{22}$$

The optimization variables $\alpha_{f,n}$, $\beta_{j,f,n}$, γ_e, and $\zeta_{j,e}$ are auxiliary variables introduced to derive the robust counterparts.

G. SFC Orchestration Problem

The SFC orchestration problem is defined as the following constrained optimization problem:

Minimize $\sum_{n \in N} P_n$

s.t. resource cstrs: (3)–(5), (19) and (20) — energy cstrs: (17) and (18)

routing cstrs:(15), (16), (21) and (22) — placement cstrs: (10)–(13)

delay cstrs: (6)–(9).

IV. Exact Orchestration Algorithm

We note that constraints (6), (8) and (9) in the above formulation are non-linear. In this section, we show how these constraints can be reformulated in order to transform the model into a MISOCP, which can be solved using standard solvers such as CPLEX [20] and Gurobi [21].

Let $\tilde{d}^{bwd}_e$ and h^{bwd}_e, repsectively, denote the upper bound on the queueing delay and the residual capacity of link e. Then (6) can be rewritten as follows:

$$\tilde{d}^{bwd}_e h^{bwd}_e \geq w_e, \ \forall e \in E, \tag{23}$$

$$h^{bwd}_e = c^{bwd}_e - r^{bwd}_e, \ \forall e \in E. \tag{24}$$

The product term in (23) can be expressed as a sum of squares as follows:

$$2w^2_e + (\tilde{d}^{bwd}_e)^2 + (h^{bwd}_e)^2 \leq (\tilde{d}^{bwd}_e + h^{bwd}_e)^2, \ \forall e \in E. \tag{25}$$

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. In the same way, we can reformulate (8) as follows:

$$2i_{f,n}^2 + (\tilde{d}_{f,n}^{proc})^2 + (h_{f,n}^{proc})^2 \leq (\tilde{d}_{f,n}^{proc} + h_{f,n}^{proc})^2, \quad \forall f \in F, \forall n \in N_{server}, \tag{26}$$

$$h_{f,n}^{proc} = c_{f,n}^{proc} - r_{f,n}^{proc}, \ \forall f \in F, \forall n \in N_{server}, \tag{27}$$

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 (9) by their upper bounds:

$$\sum_{j \in S_J} \sum_{e \in E} l_{j,e}(\tilde{d}_e^{bwd} + d_e^{prop}) + \sum_{v \in S_v} \sum_{n \in N_{server}} \sum_{\substack{f \in F \\ t(v)=f}} a_{v,n} \tilde{d}_{f,n}^{proc} \leq \theta_S, \ \forall S \in \mathcal{U}. \tag{28}$$

We use the big-M approach to linearize the products of binary and continuous variables in (28). We define $\tilde{d}_{j,e}^{bwd} = l_{j,e}\tilde{d}_e^{bwd}$ as the upper bound on the queuing delay experienced by traffic flow j on link e. Assume that $\tilde{d}_e^{bwd}$ is bounded by a constant M_1, we can express $\tilde{d}_{j,e}^{bwd} = l_{j,e}\tilde{d}_e^{bwd}$ by the following set of linear constraints:

$$\tilde{d}_e^{bwd} \leq \tilde{d}_{j,e}^{bwd} + (1 - l_{j,e})M_1, \ \forall e \in E, \forall S \in \mathcal{U}, \forall j \in S_J, \tag{29}$$

$$\tilde{d}_{j,e}^{bwd} \leq l_{j,e}M_1, \ \forall e \in E, \forall S \in \mathcal{U}, \forall j \in S_J, \tag{30}$$

$$\tilde{d}_{j,e}^{bwd} \leq \tilde{d}_e^{bwd}, \ \forall e \in E, \forall S \in \mathcal{U}, \forall j \in S_J. \tag{31}$$

Similarly, given a VNF request v of type f (*i.e.*, $t(v) = f$), 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. Let constant M_2 denote an upper bound on $\tilde{d}_{f,n}^{proc}$. We can linearly express $\tilde{d}_{v,n}^{proc} = a_{v,n}\tilde{d}_{f,n}^{proc}$ as follows:

$$\tilde{d}_{f,n}^{proc} \leq \tilde{d}_{v,n}^{proc} + (1 - a_{v,n})M_2, \quad \forall n \in N_{server}, \forall f \in F, \forall S \in \mathcal{U}, \forall v \in S_V : t(v) = f, \tag{32}$$

$$\tilde{d}_{v,n}^{proc} \leq a_{v,n}M_2, \ \forall n \in N_{server}, \forall f \in F, \forall S \in \mathcal{U}, \forall v \in S_V, \tag{33}$$

$$\tilde{d}_{v,n}^{proc} \leq \tilde{d}_{f,n}^{proc}, \ \forall n \in N_{server}, \forall f \in F, \forall S \in \mathcal{U}, \forall v \in S_V : t(v) = f. \tag{34}$$

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. Thereby, 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 a linear constraint as follows:

$$\sum_{j \in S_J} \sum_{e \in E} \tilde{d}_{j,e}^{bwd} + l_{j,e}d_e^{prop} + \sum_{v \in S_v} \sum_{n \in N_{server}} \sum_{\substack{f \in F \\ t(v)=f}} \tilde{d}_{v,n}^{proc} \leq \theta_S, \ \forall S \in \mathcal{U}. \tag{35}$$

One important consideration is the specification of the deadline θ_S. In practice, it is difficult to determine the exact value of θ_S for a given service, as it dependents on many factors including the client preference for performance and cost. While our model, in its current form, can accommodate any *hard* deadline, if we allow some controlled slack in the service delays, the algorithm can achieve substantially lower cost for orchestrating the same set of services. In the following, we present the modified algorithm (with slack) and show that it can be tuned to achieve the same hard deadlines, if so desired. Define a new variable $z_S \geq 0$ which indicates the amount of delay slack for each SFC S and replace (35) by the following:

$$\sum_{j \in S_J} \sum_{e \in E} \tilde{d}_{j,e}^{bwd} + l_{j,e}d_e^{prop} + \sum_{v \in S_v} \sum_{n \in N_{server}} \sum_{\substack{f \in F \\ t(v)=f}} \tilde{d}_{v,n}^{proc} - \theta_S \leq z_S, \quad \forall S \in \mathcal{U}. \tag{36}$$

A new term that captures the penalty for violating the deadlines is then added to the objective:

$$\text{Minimize} \ \sum_{n \in N} P_n + \lambda \sum_{S \in \mathcal{U}} z_S, \tag{37}$$

where λ is a parameter that expresses the importance of fulfilling the hard deadlines. Specifically, by setting λ to a sufficiently large value (*e.g.*, ∞), the model is forced to meet the hard deadlines. Alternatively, by setting λ to smaller values, the model can trade-off the delay performance for reduced energy cost.

Our algorithm called SODL (Sfc Orchestration with DeadLine) to solve the problem exactly using a MISOCP formulation is given in Algorithm 1.

Algorithm 1 SODL (Exact Orchestration)

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

s.t. resource cstrs: (3)–(5), (19) and (20) — energy cstrs: (17) and (18)

routing cstrs: (15), (16), (21) and (22) — placement cstrs: (10)–(13)

delay cstrs: (24)–(27), (29)–(34) and (36).

V. Approximate Orchestration Algorithm

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 [2] 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 [22]. 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. 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. Note that out of the six integer variables, o_n, w_e and $i_{f,n}$ are derived variables, which will be automatically fixed when $a_{v,n}$ and $l_{j,e}$ are fixed. This leaves us with three variables to be fixed including $a_{v,n}$, $l_{j,e}$ and $m_{f,n}$.

TABLE I: Summary of Notations

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

Algorithm 2 xSODL (Approximate Orchestration)

Input: All model's parameters, $\mathcal{M}$, 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: $\mathcal{M}.updateConstr((5), \sum_{f \in F} m_{f,n} \leq M_n - |\mathcal{F}_n| + 1)$ ▷ 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`$(\mathcal{M},\ n_1,\ n_2,\ j,\ \mathcal{L})$
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: $\mathcal{M}.updateConstr((5), \sum_{f \in F} m_{f,n} \leq M_n)$ ▷ Restore servers' capacities
38: $\mathcal{M}_{sol} \leftarrow$ Solves the relaxed program
39: **return** $\mathcal{M}_{sol}$

A. VNF placement

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 VI, the reduction in runtime achieved by increasing Υ comes at the higher energy provisioning cost.

Solution Feasibility. Since each server node in the substrate network is constrained by its capacity, 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 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 at server n (see subsection V-C).

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.

In both cases, `RestrictPlacement` updates N_v to contain only the mappings that do not make the model infeasible. For mappings eliminated due to the capacity limitation in Case (I), `RestrictPlacement` also adds additional constraints to prevent the model in the next run (line 24) from mapping fractions of virtual nodes to servers that do not have enough capacity to support the full assignment.

B. Routing

Fractional Routing. The mapping of virtual links (flows) to physical links in the substrate network 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 subflows 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 [23]. 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.

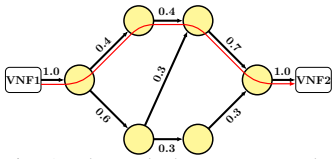


Fig. 1: The path that can carry the largest fractional amount of flow is the one that is most likely to cost the least energy and delay.

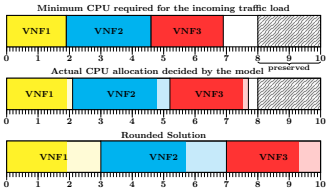


Fig. 2: Reserving 2 CPUs on a server with 3 VNF instances ensures that the rounding does not violate the capacity.

Solution 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. To assess whether physical link e has enough capacity to map virtual link j, the algorithm keeps a record of the virtual links that have been mapped to e (*i.e.*, when performing routing in `Route`). Then, assuming that j is already mapped to e, `RestrictRouting` calculates the total bandwidth demand on e, and verifies that it is less than the link's capacity.

C. CPU allocation

Fractional 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 $j_{j,e}$, $m_{f,n}$ 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. 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).

Solution 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 on the server (*i.e.*, not the number of virtual nodes). Given that there are $|\mathcal{F}_n|$ VNF instances on server n, we show that by reserving $|\mathcal{F}_n| - 1$ CPU cores, the capacity of the server will not be violated due to rounding. We have,

$$\begin{aligned}
&\sum_{f\in\mathcal{F}_n} m_{f,n} \le M_n - (|\mathcal{F}_n| - 1)\\
&\implies m_{f_x,n} + \sum_{f\in\mathcal{F}_n-\{f_x\}} \lceil m_{f,n}\rceil \le M_n - (|\mathcal{F}_n| - 1) + (|\mathcal{F}_n| - 1)\\
&\iff m_{f_x,n} + \sum_{f\in\mathcal{F}_n-\{f_x\}} \lceil m_{f,n}\rceil \le M_n\\
&\iff \sum_{f\in\mathcal{F}_n} \lceil m_{f,n}\rceil \le M_n.
\end{aligned}$$

Once the algorithm has finished fixing $m_{f,n}$, it restores the capacities of the servers (line 37) and reruns the model (line 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. Performance Evaluation

We conduct extensive simulations to assess the performance of our exact (*i.e.*, SODL) and approximate (*i.e.*, xSODL) algorithms in terms of being able to proactively deal with demand fluctuations as well as scaling to large problem instances. Both algorithms are implemented in Gurobi [21] and run on an Intel Core i7-3720QM@2.6GHz machine with 16 GB RAM.

TABLE II: Settings for virtual network functions (from [24]).

VNF f	Abbr.	Proc. 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

TABLE III: Settings for servers and switches (from [25], [26]).

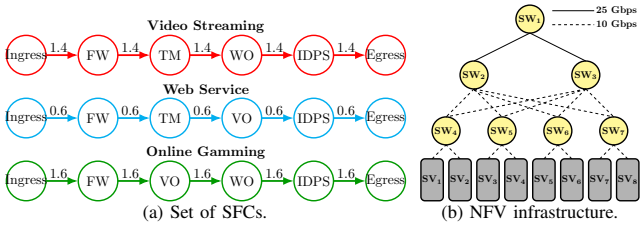
Server configurations (4 classes)				
No. of cores (M_n)	4 cores	6 cores	8 cores	10 cores
Raw core capacity (c_n^{cpu})	1.0 Gbps	1 Gbps	1.0 Gbps	1 Gbps
Idle power (P_n^{active})	70 Watts	75 Watts	80 Watts	100 Watts
Max power (P_n^{max})	200 Watts	250 Watts	300 Watts	350 Watts
Switch configurations				
Switching capacity (c_n^{bwd})	120.0 Gbps			
Idle power (P_n^{active})	30 Watts			
Max power (P_n^{max})	60 Watts			

A. Exact Algorithm 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 [27]). The 12 VNFs in the service chains belong to five different VNF types. The processing coefficient for each VNF type is specified in Table II. We assume that 3 licenses for each VNF type are available. By default, the deadlines for all SFCs are set to 0.2 milliseconds [28]. These deadlines 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 (*i.e.*, 18 full-duplex connections) 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 configurations for servers and switches.

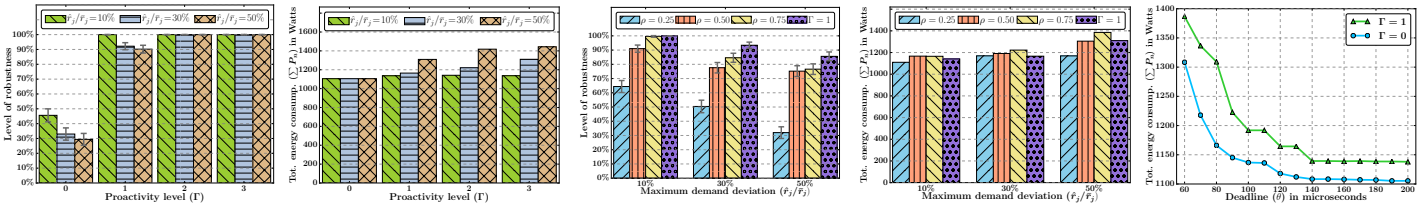
Baseline Algorithm. We compare SODL with a *baseline* algorithm, which is constructed by adding a *safety margin* to each SFC's nominal demand to account for future demand fluctuations. Specifically, if the demand profile of flow j is given by $[\bar{r}_j - \hat{r}_j, \bar{r}_j + \hat{r}_j]$, then the baseline algorithm assigns the demand $\bar{r}_j + \rho\hat{r}_j$ to j, for some $0 \le \rho \le 1$. Clearly, a range of proactive orchestrations can be achieved by changing the value of ρ. 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.

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 Γ, which is the parameter that controls the proactivity of our algorithms. 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



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

Fig. 3: Exact model evaluation settings.



(a) Effect of demand fluctuations. (b) Price of robustness. (c) Relative protection levels. (d) Relative energy cost. (e) Effect of deadlines.

Fig. 4: Performance of SODL under different levels of proactivity, demand fluctuations and deadlines.

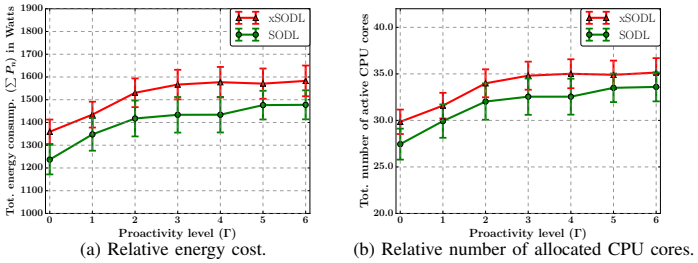
corresponding proactive placement. Then, we randomly generate 500 demand vectors (consistent with the demand profile), where each vector consists of 3 demands, one for each deployed SFC. For each demand vector, we check whether the demands can be satisfied by the computed placement. This allows us to compute the percentage of feasible (infeasible) demands, which indicates how proactive (reactive) the orchestration algorithm is. 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 1 makes the computed placement 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 produces placements provision more network resources to cope with higher future demand fluctuations, thus increasing the energy consumption. The effect of Γ on the energy consumption becomes even more pronounced as demands exhibit a larger range of variation.

Comparison with Baseline. 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. 4(c) and 4(d), 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 placement 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 $\hat{r}_j/\bar{r}_j$.

Effect of Deadline. Fig. 4(e) illustrates the effect of the deadline θ on energy consumption for $\Gamma = 0$ and $\Gamma = 1$. When the deadline is tight, we observe that a small change in the deadline drastically changes the energy consumption. However, as the deadline becomes loose, the effect of its change on the energy consumption becomes less significant. This is anticipated since unless there is a tight deadline, the model only needs to provision just enough resources to satisfy the demands.

Effect of Approximations. The objective of this experiment is to compare the performance of SODL and xSODL. We run both algorithms on 100 demand profiles with different proactivity level Γ and record the average energy consumption and number of allocated CPU cores to all VNF instances. Each demand profile is specified by the relative maximum deviation $\hat{r}_j/\bar{r}_j = 20\%$, where the nominal demand is generated randomly over the range $[0.5, 2.5]$ (Gbps).

The results are, respectively, presented in Figs. 5(a) and 5(b) with the 95% confidence intervals. We can see that the placement solution computed by xSODL consumes about 10% (100 Watts) more energy than that of SODL. This difference is less than the maximum energy consumption of a single server, which implies that SODL and xSODL generally agree on the number of servers to deploy. The recorded numbers for allocated CPU cores shown in Fig. 4(d) also confirm this conclusion. Recall that the minimum capacity of a server is 4 CPU cores. We see that xSODL allocates only 2 to 3 more cores than SODL.



(a) Relative energy cost. (b) Relative number of allocated CPU cores.

Fig. 5: Comparison of SODL and xSODL.

B. Approximate Algorithm Evaluation

Simulation Settings. To asses the performance of xSODL, we consider a scaled-up version of the scenario considered for SODL evaluation. 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. Demand profiles in this experiment are specified by the relative maximum deviation $\hat{r}_j/\bar{r}_j = 20\%$, where the nominal demands are generated randomly over the range $[0.5, 2.5]$ (Gbps). The NFVI is still a Clos data center topology as in Fig. 3(b), but has a larger capacity with 24 servers, 21 switches and 132 directed links. All other settings remain the same.

Runtime-Energy Trade-off. In this experiment, we run xSODL on the randomly generated demand profiles, and record the average energy consumption and runtime for each value of Υ. Recall that Υ is the number of VNFs fixed in each iteration of the model. The results along with the 95% interval are reported in Fig. 6. As expected, fixing more virtual nodes per iteration reduces the time it takes to run the algorithm. Though this benefit comes at a price. As Υ increases, the likelihood of virtual nodes being assigned to unfavourable locations, which incur more energy cost, increases since more fractional assignment solutions are selected to be rounded.

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

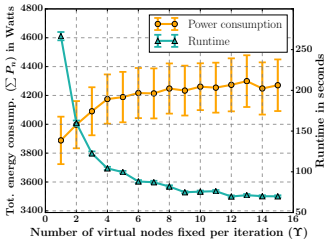


Fig. 6: Runtime-energy trade-off in xSODL.

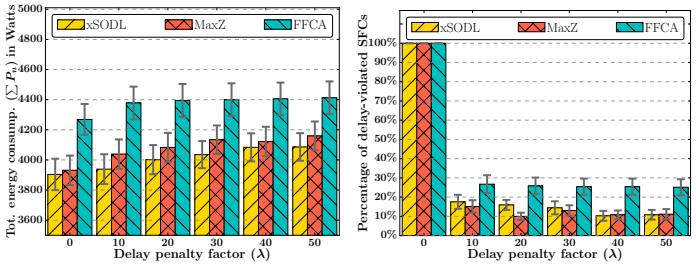
- First Fit Clustering Allocation (FFCA) [1]: In this algorithm, servers connecting to the same switch form a cluster. FFCA tries to allocate all VNF requests on the same SFC to servers within a cluster, with the server that has the most amount of resources being considered first.
- MaxZ [2]: Similar to xSODL, MaxZ employs iterative rounding to decide where to map the VNF requests. However, while xSODL performs capacity checks and adds additional constraints to prevent the model from fractionally assigning VNF requests to servers that do not have enough capacities in the next run, MaxZ does not prevent such cases from happening. After each run of the relaxed model, MaxZ simply 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 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 services (*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.

VII. Conclusion

In this work, we considered the SFC orchestration problem with the objective of minimizing the energy usage of the NFVI, while respecting the deadline for each service. We developed exact and approximate algorithms to compute proactive orchestration solutions that can cope with fluctuations in future service demands. The proposed algorithms capture all key aspects of SFC orchestration, including dynamic CPU allocation based on demand, sharing VNF instances among multiple service flows and dependence of delay on traffic load. Extensive simulations demonstrate that explicitly considering demand fluctuations results in more proactive and efficient resource allocation. An interesting future work is to extend our model to include more information about future demands, beyond the range of fluctuations currently considered.



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

Fig. 7: Comparing xSODL with existing algorithms.

References

[1] 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.

[2] S. Agarwal *et al.*, "Joint VNF placement and CPU allocation in 5G," in *IEEE INFOCOM*, 2018.

[3] M. Chiosi *et al.*, "Network functions virtualization: An introduction, benefits, enablers, challenges & call for action," *ETSI White Paper*, 2012.

[4] ETSI, "Network function virtualization(NFV); management and orchestration," *ETSI GS NFV-MAN*, 2014.

[5] H. Ballani *et al.*, "Towards predictable datacenter networks," in *ACM SIGCOMM Comput. Commun. Rev.*, vol. 41, no. 4, 2011.

[6] J. G. Herrera and J. F. Botero, "Resource allocation in NFV: A comprehensive survey," *IEEE Trans. Netw. Service Manag.*, vol. 13, no. 3, 2016.

[7] B. Yi *et al.*, "A comprehensive survey of network function virtualization," *Computer Networks*, vol. 133, 2018.

[8] X. Wang *et al.*, "Online VNF scaling in datacenters," in *IEEE CLOUD*, 2016.

[9] B. Zhang, J. Hwang, and T. Wood, "Toward online virtual network function placement in software defined networks," in *IEEE/ACM IWQoS*, 2016.

[10] X. Wang *et al.*, "Online learning-assisted VNF service chain scaling with network uncertainties," in *IEEE CLOUD*, 2017.

[11] X. Zhang *et al.*, "Proactive VNF provisioning with multi-timescale cloud resources: Fusing online learning and online optimization," in *IEEE INFOCOM*, 2017.

[12] M. Shit *et al.*, "Competitive online convex optimization with switching costs and ramp constraints," in *IEEE INFOCOM*, 2018.

[13] V. S. Reddy, A. Baumgartner, and T. Bauschert, "Robust embedding of VNF/service chains with delay bounds," *IEEE NFV-SDN*, 2016.

[14] A. Marotta and A. Kassler, "A power efficient and robust virtual network functions placement problem," in *IEEE ITC*, 2016.

[15] 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.

[16] M. Nguyen, M. Dolati, and M. Ghaderi, "Proactive service orchestration with deadline," Tech. Rep. [Online]. Available: https://pages.cpsc.ucalgary.ca/~minh.nguyen5/techrepVNF.pdf

[17] B. Yi, X. Wang, and M. Huang, "A generalized VNF sharing approach for service scheduling," *IEEE Commun. Lett.*, vol. 22, no. 1, 2018.

[18] M. Dayarathna, Y. Wen, and R. Fan, "Data center energy consumption modeling: A survey," *IEEE Commun. Surveys Tuts.*, vol. 18, no. 1, 2016.

[19] D. Bertsimas, D. B. Brown, and C. Caramanis, "Theory and applications of robust optimization," *SIAM Review*, vol. 53, no. 3, 2011.

[20] CPLEX. (2018). [Online]. Available: https://www.ibm.com/ca-en/marketplace/ibm-ilog-cplex

[21] Gurobi. (2018). [Online]. Available: http://www.gurobi.com/

[22] 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.

[23] 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.

[24] J. Martins *et al.*, "ClickOS and the art of network function virtualization," in *USENIX NSDI*, 2014.

[25] Cisco. (2018) Power Calculator. [Online]. Available: http://ucspowercalc.cisco.com

[26] Aruba. (2018) 3810 Switch Series. [Online]. Available: https://www.arubanetworks.com/assets/ds/DS_3810SwitchSeries.pdf

[27] N. Huin, B. Jaumard, and F. Giroire, "Optimal network service chain provisioning," *IEEE/ACM Trans. Netw.*, 2018.

[28] D. A. Popescu, N. Zilberman, and A. W. Moore, "Characterizing the impact of network latency on cloud-based applications performance," University of Cambridge, Tech. Rep., 2017.