Divide and Conquer: Hierarchical Network and Service Coordination

Stefan Schneider
Paderborn University, Germany
stefan.schneider@upb.de

Mirko Jürgens
Paderborn University, Germany
mirkoj@mail.upb.de

Holger Karl
Paderborn University, Germany
holger.karl@upb.de

***Abstract*—In practical, large-scale networks, services are requested by users across the globe, e.g., for video streaming. Services consist of multiple interconnected components such as microservices in a service mesh. Coordinating these services requires scaling them according to continuously changing user demand, deploying instances at the edge close to their users, and routing traffic efficiently between users and connected instances. Network and service coordination is commonly addressed through centralized approaches, where a single coordinator knows everything and coordinates the entire network globally. While such centralized approaches can reach global optima, they do not scale to large, realistic networks. In contrast, distributed approaches scale well, but sacrifice solution quality due to their limited scope of knowledge and coordination decisions.**

To this end, we propose a hierarchical coordination approach that combines the good solution quality of centralized approaches with the scalability of distributed approaches. In doing so, we divide the network into multiple hierarchical domains and optimize coordination in a top-down manner. We compare our hierarchical with a centralized approach in an extensive evaluation on a real-world network topology. Our results indicate that hierarchical coordination can find close-to-optimal solutions in a fraction of the runtime of centralized approaches.

I. Introduction

Modern services, e.g., for video streaming, consist of multiple interconnected components; this holds for examples like microservices in a service mesh [1] or virtual network functions (VNFs) in a network service [2]. Users from around the globe may request these services and expect short delays and high service quality. At the same time, operators are looking for ways to lower their capital and operational expenditures.

Hence, services and their components need to be scaled according to the current load such that user demand is satisfied yet resources are not wasted. Instances of these components can be deployed on any compute node in the network. Instances deployed closer to users reduce end-to-end delay and improve service quality. However, deploying more instances increases costs (e.g., due to licensing and/or resource consumption). User demand needs to be balanced among multiple available instances and traffic needs to be routed between the users and interconnected instances. In doing so, limited compute and link capacities need to be respected and the number of instances as well as path delays should be minimized. Overall, optimized network and service coordination (including scaling, placement, and routing) is crucial for both users and operators to ensure high service quality and low costs. Simultaneously, this is challenging due to many influencing factors such as limited resources, trade-offs between objectives, and continuously changing demand.

Even taking just some of these factors into account leads to optimization problems that are NP-hard [3]. Even with heuristic, non-optimal approaches, practical scalability is usually limited to small networks. A natural approach is hence to split large networks into smaller domains, which are then coordinated independently in a distributed fashion. While this provides reasonable results within shorter runtimes, solution quality may be sacrificed due to lack of global knowledge and necessarily sub-optimal inter-domain coordination.

We propose a novel hierarchical coordination approach where we combine benefits of both centralized and distributed approaches. Our approach follows a divide-and-conquer strategy where the network can be divided into arbitrary levels of hierarchy. First, in a bottom-up phase, lower-level hierarchies aggregate and advertise information to the higher-level hierarchies (cf. "one big switch" abstraction in SDN [4]). In the following top-down phase, high-level coordinators make inter-domain coordination decisions based on the advertised information. All child coordinators then refine the coarse-grained decisions of their parents in parallel.

A challenge here is that node and link capacity constraints must be respected on all hierarchies. Decisions of higher-level coordinators should guide child coordinators by reducing their decision space but must not keep them from finding a valid solution if any feasible solution exists. If no feasible solution exists, e.g., because resources are already highly utilized, this should be noticed at the top hierarchies to quickly reject the corresponding service requests without the unnecessary overhead of recursing to lower hierarchies. The advertised information (e.g., about available resources) needs to be detailed and relevant enough to enable meaningful decisions at high-level hierarchies. Still, it should be aggregated and abstract enough to reduce coordination complexity on higher levels.

To navigate this trade-off, we define a suitable information

978-3-903176-32-4 © 2021 IFIP

advertisement scheme. We then formalize our hierarchical coordination approach as mixed integer linear program (MILP) and optimize it numerically. In our extensive evaluation on a real-world network topology, we compare our hierarchical approach with an equivalent centralized approach. We find that our approach achieves comparable solution quality but is more than 10x faster on average. *Overall, our contributions are:*

- We propose a generic coordination approach that works with any given hierarchical structure (Sec. IV).
- We formalize an MILP that is solved numerically by coordinators on each hierarchical level (Sec. V).
- We evaluate our approach on a real-world network topology comparing it against a centralized approach (Sec. VI).
- For reusability, our implementation is open source [5].

II. Related Work

Considerable effort has been focused on network and service coordination in the context of network function virtualization (NFV) [6] or cloud or edge computing [7], [8]. However, most authors propose centralized approaches to make global decisions for coordinating services in the entire network [9]–[16]. These approaches typically only work in small networks and do not scale to practical large-scale networks [11]. Distributed approaches [17], [18] are scalable but lack coordination between domains. To the best of our knowledge, we propose the first hierarchical approach for network and service coordination, which solves these inherent limitations. By dividing the network into hierarchies, they can be optimized in a scalable and distributed, yet coordinated fashion.

Nevertheless, authors have proposed hierarchical approaches in related areas [19], [20]. In particular, virtual network embedding (VNE) is well-studied [21] and closely related to our problem. Samuel et al. [22] propose a distributed and hierarchical approach that greedily solves the VNE problem while different domains hide as much information from each other as possible because they belong to competitors. While this perspective is interesting, we assume domains to cooperate and share relevant topology information to enable optimized coordination. Due to limited information sharing and its greedy nature, the approach by Samuel et al. easily gets stuck at suboptimal solutions or fails to find any feasible solution. In our approach, we ensure that any solution suggested by the top hierarchy can be refined into a feasible embedding.

Similar to us, Ghazar et al. [23] assume that domains cooperate. However, the authors do not aggregate or abstract any information such that computations on high hierarchical levels become very expensive, comparable to centralized approaches. Hence, the authors skip higher levels and directly select an intermediate hierarchical level to embed the full request in one of its sub-domains. If this is not possible because the domains are too small, the process is repeated on the next higher level. This leads to considerable overhead for large embeddings, e.g., where ingress and egress are far apart. Our approach efficiently coordinates across domains by aggregating and simplifying information on higher hierarchical levels.

In general, our network and service coordination problem is considerably more complex than VNE. Unlike typical VNE approaches, we consider routing from ingress to egress but also dynamic reuse of components across services and dynamic scaling, i.e., flexible number and resources per instance. Such joint coordination is important to successfully balance trade-offs [3], [34], [35]. Overall, there is a stronger interdependence between decisions of different hierarchical levels and domains, making the problem more challenging than VNE.

Finally, hierarchical approaches are common in traffic engineering, which is a subproblem of network and service coordination. For example, MPLS uses a path computation element (PCE) to find shortest paths across different hierarchical domains (AS) [24]–[26]. Similar to our approach, topology information from these different domains can be abstracted and aggregated through topology aggregation mechanisms [27]. Related to our aggregation approach, Secci et al. [28] propose a full mesh aggregation containing the most relevant topology information. Still, the authors assume that advertised intra-domain paths do not overlap, i.e., do not share links. In contrast, our approach supports overlapping intra-domain paths and explicitly advertises constraints to avoid overloading shared links. Hence, our approach allows more paths to be advertised such that better embeddings can be found.

III. Problem Statement

We address the network and service coordination problem where users request services in a network of distributed nodes. Accordingly, the chained components of a service need to be scaled and instantiated on network nodes and traffic routed through them, connecting users and instances.

We model the network as graph $G = (V, L)$ with nodes V and links L. Each node $v \in V$ has a compute capacity $\kappa_v^{\text{cap}} \in \mathbb{R}_{\geq 0}$ (e.g., CPU)[1]. Each link $l \in L$ connects two nodes bidirectionally with a maximum data rate $\lambda_l^{\text{cap}} \in \mathbb{R}_{\geq 0}$ (shared in both directions) and delay $d_l \in \mathbb{R}_{\geq 0}$. Users in the network request services, where each request $r = (s_r, v_r^{\text{in}}, v_r^{\text{eg}}, \lambda_r) \in R$ is defined by the requested service s_r, ingress node v_r^{in}, egress node v_r^{eg}, and required data rate λ_r. We assume that a request's traffic can be split over multiple paths from ingress to egress. For unsplittable traffic, additional constraints could be added to our MILP [29]. Let V^{in} be the set of all ingress and V^{eg} the set of all egress nodes ($V^{\text{in}}, V^{\text{eg}} \subseteq V$).

A service $s \in S$ is defined by its chain of components $C_s = \langle c_s^1, ..., c_s^{m_s} \rangle$, each providing parts of the service functionality (e.g., security, compression, optimization, ...). A component c may be reused across different services. We denote the set of all components from all services as C. Each component $c \in C$ can be scaled flexibly and instantiated zero, one, or more times

[1]This generic compute capacity can easily be extended to multiple different resource types, e.g., GPU, memory, and storage, by using a vector instead.

across different nodes in the network. Requests need to traverse instances of all service components in the specified order.

An instance can process multiple requests in parallel, possibly belonging to different services. In doing so, it requires resources proportional to the total data rate it is processing. In particular, we model resource requirements as linear function $\kappa_c(\lambda) = \alpha_c \cdot \lambda$ of the total traversing data rate λ. All instances of a component c have the same component-specific coefficient α_c. Furthermore, components may augment or compress traversing data affecting the data rate (e.g. WAN optimizers) [30]. Function $\mu_c(\lambda) = \beta_c \cdot \lambda$ defines the outgoing data rate for instances of component c, based on the total traversing data rate λ and coefficient β_c. While such linear functions are a fairly accurate representation of real-world component characteristics [31], [32], the model can easily be extended to more flexible piece-wise linear functions [33].

We adopt the perspective of serverless computing and focus on inter-node coordination. When instantiating a component c on node v, we assume that *within* node v (intra-node) a system like Kubernetes [36] or an operating system transparently deploys c on the node's internal resources (machines, cores).

IV. Hierarchical Coordination Approach

The main idea of our approach is to divide the network into smaller domains and coordinate them in a hierarchical manner. Each domain is a part of the network that may recursively consist of sub-domains, forming a hierarchy. This hierarchical approach allows both efficient parallel coordination of different domains yet necessitates coordination between domains for highly optimized results. We assume that dividing the network into hierarchies of domains and sub-domains is out of scope and happens before coordination starts, e.g., based on node locality or business aspects. Our approach is not tied to any structure and works with any given domains and hierarchies.

Given domains and hierarchies, our approach consists of two phases: First, domains aggregate and advertise relevant information (e.g., about available resources) to their coordinators in a bottom-up manner. Second, based on this information, the coordinators make coordination decisions in a top-down manner. We choose top-down coordination to allow high-level coordinators to optimize inter-domain decisions and guide lower-level coordinators. Starting coordination directly at a lower level would often lead to worse solutions. We ensure that each high-level coordination decision can be further refined into a feasible solution or directly reject requests at the top level. Hence, we avoid overhead of jumping up and down between levels to backtrack and fix infeasible embeddings. To enable efficient top-down coordination, a main challenge is advertising relevant but aggregated information from lower levels in phase 1. More detailed information allows higher quality coordination but also increases complexity. In the following, we introduce our notation for domains and hierarchies and describe the two phases in more detail (see Alg. 1).

Algorithm 1 Hierarchical Coordination Algorithm

```
1: for $k = 1$ up to $\hat{k} - 1$ do                                  ▷ Phase 1
2:     for $i \in \{1, ..., n_{k-1}\}$ in parallel do
3:         Aggregate sub-domain information as $\bar{D}_i^{k-1}$
4:     Advertise $\bar{D}_j^k = \{\bar{D}_i^{k-1} | \forall i\}$ to coord. $j$ on level $k+1$
5: for $k = \hat{k}$ down to 1 do                                        ▷ Phase 2
6:     for $j \in \{1, ..., n_k\}$ in parallel do
7:         Embed request $r_j^k$ into $\bar{D}_j^{k-1}$ by solving the MILP
8:         Split request $r_j^k$ into $r_i^{k-1}$ for all coord. $i$ on $k-1$
```

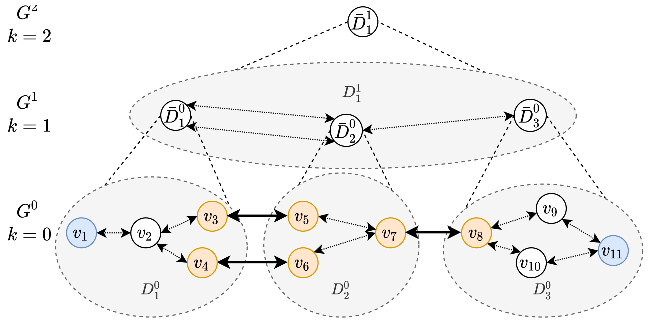


Fig. 1: Example with $\hat{k} = 2$ hierarchies. Ingress and egress nodes are shown in blue and border nodes in orange.

A. Domains and Hierarchies

We denote the total number of hierarchical levels as $\hat{k}$ and a specific level as $k \leq \hat{k} \in \mathbb{N}_0$, where $k = 0$ is the substrate network $G = G^0$. In the example of Fig. 1, the substrate network $G^0 = (V^0, L^0)$ is split into $n_0 = 3$ separate domains D_1^0, D_2^0, D_3^0 with $D_i^0 = (V_i^0, L_i^0)$. Each domain D_i^0 is coordinated separately by its coordinator on $k = 1$, in parallel with the other domains D_j^0. At level $k = 1$, nodes are grouped again into domains that are handled by coordinators on $k = 2$ (a single domain D_1^1 in Fig. 1). This definition recursively extends to an arbitrary number of $\hat{k}$ hierarchies.

While we assume that all nodes V^k on level k belong to some domain D_i^k (i.e., $V^k = \bigcup_{i=1}^{n_k} V_i^k$), not all links L^k are part of some domain. In particular, we distinguish between intra-domain and inter-domain links. Intra-domain links L_i^k connect nodes within a single domain D_i^k (lighter in Fig. 1). Inter-domain links do not belong to any domain but connect nodes across two different domains (thicker in Fig. 1). We define border nodes $B_i^k \subseteq V_i^k$ as the subset of nodes that have an inter-domain link to another domain (orange in Fig. 1). For example in Fig. 1, $B_2^0 = \{v_5, v_6, v_7\}$.

B. Bottom-Up Information Advertisement (Phase 1)

Each domain's coordinator scales and places services as well as routes traffic inside the domain. It needs to know about available compute capacity, data rate limitations, and delays within the domain. A domain on level k may comprise multiple

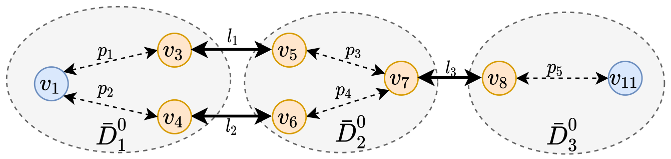


Fig. 2: $\bar{D}_1^1$ advertised to the coordinator on $k = 2$ in Fig. 1.

levels of sub-domains and cover large parts of the network. Thus, it is crucial to hide unnecessary information of lower levels from higher-level coordinators to reduce complexity and ensure scalability. Hence, domains aggregate and advertise relevant information of their sub-domains to their coordinators as follows (ln. 1–4 in Alg. 1).

For a sub-domain D_i^k domain information $\bar{D}_i^k = (\mathcal{V}_i^k, \mathcal{P}_i^k, \mathcal{L}_i^k)$ is advertised to the coordinator. Fig. 2 illustrates the advertised domain $\bar{D}_1^1$ to the coordinator on $k = 2$, containing aggregated information about sub-domains D_1^0–D_3^0 from Fig. 1. First, the advertised information includes a subset $\mathcal{V}_i^k \subseteq V_i^k$ of the domain's nodes. Subset $\mathcal{V}_i^k = \{V^{\text{in}} \cap V_i^k\} \cup \{V^{\text{eg}} \cap V_i^k\} \cup B_i^k$ includes all ingress and egress nodes within V_i^k as well as the domain's border nodes but no intermediate nodes. The advertised network in Fig. 2 includes ingress v_1 and egress v_{11} as well as border nodes v_3–v_8 but not intermediate nodes v_2, v_9, and v_{10}. In this small example, the majority of nodes is advertised to the coordinator. However, in larger networks with more intermediate nodes and additional levels of sub-domains, more nodes would be hidden and excluded from $\mathcal{V}_i^k$ to ensure efficient coordination.

In addition to nodes $\mathcal{V}_i^k$, domain D_i^k also advertises information about its intra-domain paths $\mathcal{P}_i^k$ and inter-domain links $\mathcal{L}_i^k$. Paths in $\mathcal{P}_i^k$ indicate connections between nodes inside a domain, possibly via multiple hops. For example in Fig. 2, v_1 can reach v_3 via intermediate, hidden node v_2 and thus has a path p_1 to v_3. Links $\mathcal{L}_i^k$ are inter-domain links between D_i^k and another domain.

To support coordination, each intra-domain path $p \in \mathcal{P}_i^k$ is annotated with further information about available compute capacity, data rate limitations, and delay along the path. This is necessary for coordinators to decide where to place instances and how to route traffic within the domain. The coordinator needs to know how much traffic (with which data rate) and at which delay can be routed through the domain. Traffic may arrive from a neighboring domain or originate at an ingress node inside the domain. The destination may be another neighboring domain or an egress node inside the domain. To this end, paths are calculated between all ingress, egress, and border nodes of a domain (i.e., $\mathcal{V}_i^k$) by solving the corresponding maximum flow problem. We use the Ford-Fulkerson algorithm [37] with Edmon Karp path selection [38] to find paths with maximal data rate between nodes in $\mathcal{V}_i^k$.

Why advertise complex, annotated paths and not just the domain's total data rate and compute resources? Such a naive approach would fail to inform the coordinator about path delay and possible bottlenecks, which could result in embeddings that can no longer be refined into feasible solutions. Hence, we set a path's data rate limit λ_p^{cap} to the data rate of the bottleneck link on the path. Since paths may partially overlap (e.g., p_1, p_2 in Fig. 2), P_p^λ denotes the set of all paths (including p) traversing the bottleneck link and sharing its data rate. The path's delay d_p correspond to the sum of link delays along the path. We do not consider queuing delays, but they could be added based on the current load along the path. Similarly, compute resources are advertised as properties of a *path* rather than of individual nodes since intermediate nodes of sub-domains are hidden from the coordinator. Path p's compute capacity κ_p^{cap} is the sum of compute capacities of all nodes along the path, including the source and destination node. To avoid that partially overlapping paths overload shared compute resources, we split $\kappa_p^{\text{cap}} = \kappa_p^{\text{excl}} + \kappa_p^{\text{pool}}$ into a first part of resources that are exclusively used by p and a second part that is shared with other overlapping paths. Shared resources κ_p^{pool} constitute a pool of resources shared among all paths in P_p^κ (including p). If p does not overlap with any other path, $P_p^\kappa = \{p\}$, $\kappa_p^{\text{pool}} = 0$, and $\kappa_p^{\text{cap}} = \kappa_p^{\text{excl}}$. For domains at level $k \geq 2$, the approach works similarly based on the properties of advertised paths from the domains at $k-1$.

C. Top-Down Coordination Decisions (Phase 2)

In phase 2 (ln. 5–8 in Alg. 1), the advertised information from phase 1 is used for coordination as described in the following. Phase 2 starts at the highest hierarchical level $\hat{k}$ and works top-down, where each coordinator optimizes coordination in its own domain using our MILP formulation (Sec. V). A coordinator on level $k+1$ only knows the advertised information $\bar{D}_i^k = (\mathcal{V}_i^k, \mathcal{P}_i^k, \mathcal{L}_i^k)$. Hence, the coordinator scales and places services and routes traffic on the advertised *paths* rather than directly on substrate nodes or links.

For example, assume the top-level coordinator on $\hat{k} = 2$ in Fig. 1 needs to handle a request for a service consisting of two components $C_s = \langle c_1, c_2 \rangle$ with ingress v_1 and egress v_{11}. The coordinator only knows the advertised domain $\bar{D}_1^1$ shown in Fig. 2. Based on this information, it may decide to place an instance of c_1 on path p_1 close to the ingress and an instance of c_2 on p_5 close to the egress. It could then route the traffic from v_1 in D_1^0 through D_2^0 to v_{11} in D_3^0 along p_1, l_1, p_3, l_3, p_5.

This coordination decision on $k = 2$ then needs to be refined by coordinators on $k-1$. These child coordinators decide the specific scaling, placement, and routing within each domain (D_1^0, D_2^0, D_3^0 in the example), again by solving the MILP. The parent coordinator has to ensure that its decisions are feasible and can lead to valid solutions. To build on and refine the decisions of the parent coordinator, the original request r_j^k handled on level k is adjusted and split into separate requests r_i^{k-1} for all child coordinators on $k-1$. In the aforementioned example, c_1 was placed on p_1 and

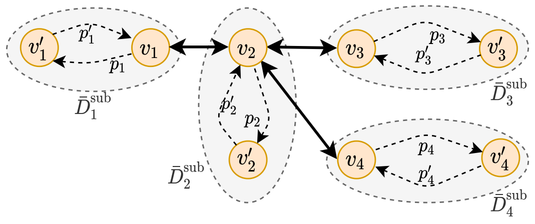


Fig. 3: $\bar{D}_1^0$ advertised to the coordinator on $k = 1$ in Fig. 1.

traffic routed from ingress v_1 via p_1, v_3, l_1 to domain D_2^0. Hence, the child coordinator of D_1^0 on $k = 1$ would receive a request for a service consisting just of $C_s = \langle c_1 \rangle$ with ingress node v_1 and egress v_3. Similarly, the coordinator of D_2^0 would receive a request for routing traffic from v_5 to v_7 (without any placement) and the coordinator of D_3^0 for placing c_2 with ingress v_8 and egress v_{11}.

To enable pure routing requests without any placement, we augment all services with auxiliary ingress and egress components $c_{\text{in}}, c_{\text{eg}} \in C$. These components are added at the front and end of a service chain, respectively, and do neither consume resources ($\kappa_c(\lambda) = 0$) nor alter traversing traffic ($\mu_c(\lambda) = \lambda$). Hence, a pure routing request would require an empty component chain $C_s = \langle c_{\text{in}}, c_{\text{eg}} \rangle$.

Coordinators on $k = 1$ constitute a special case since they directly coordinate on parts of the substrate network rather than any further sub-domains. To enable the same consistent coordination workflow with the same MILP used for $k > 1$, we automatically generate advertised domains $\bar{D}_i^0$ for substrate-level domains D_i^0 as follows. For each substrate node $v_j \in V_i^0$, we advertise a separate sub-domain $\bar{D}_j^{\text{sub}}$ containing v_j, an additional dummy node v'_j, and intra-domain paths p_j, p'_j between the two nodes. Fig. 3 shows how domain D_1^0 on Fig. 1 would be advertised as $\bar{D}_1^0$ to its coordinator on $k = 1$. The intra-domain paths are annotated with compute capacity equal to the substrate node's capacity $\kappa_{v_j}^{\text{cap}}$ and have unlimited data rate and zero delay. In doing so, coordinators on $k = 1$ can scale, place, and route on these advertised paths, similar to coordinators on $k > 1$. Coordination decisions on $k = 1$ are then transparently and automatically mapped to placement solutions on the real substrate network, where any instances placed on paths p_j, p'_j are mapped to and deployed on corresponding substrate node v_j.

Following this top-down coordination approach, decisions by high-level coordinators are further refined by child coordinators. Child coordinators solve the MILP (Sec. V) in parallel, improving response time, while the parent coordinator's decisions ensure proper coordination between domains.

V. Mixed Integer Linear Program

We formalize the mixed integer linear program (MILP) that coordinators on each level k solve for each domain D_i^k based on advertised information $\bar{D}_i^k = (\mathcal{V}_i^k, \mathcal{P}_i^k, \mathcal{L}_i^k)$. We summarize our notation in Tables I and II. The scaling and placement-related variables in the first part of Table II are 0 if the placement (i.e., an instance of component c at path p) does not exist. Similarly, the routing-related variables in the second part of the table are only defined for components c, c' where c' is a direct successor of c in the service function chain requested in r. Otherwise, the corresponding variable is 0. Compared to typical MILPs, we here need additional constraints for approximating and bounding resources and data rates and for routing based on the abstract, advertised paths $\mathcal{P}_i^k$.

TABLE I: Parameters

Symbol	Definition
κ_v^{cap}	Compute capacity of node v
$\lambda_l^{\text{cap}}, d_l$	Maximum data rate and delay of link l
$r = (s_r, v_r^{\text{in}}, v_r^{\text{eg}}, \lambda_r)$	Request $r \in R$ for service s_r, from ingress v_r^{in} to egress v_r^{out} with data rate λ_r
$C_s = \langle c_s^1, ..., c_s^{m_s} \rangle$	Chain of m_s components constituting service s
$\kappa_c(\lambda), \mu_c(\lambda)$	Resource requirements and outgoing data rate for instances of c based on ingoing data rate λ
$0 \leq k \leq \hat{k}$	Hierarchy k and top-level hierarchy $\hat{k}$
$D_i^k = (V_i^k, L_i^k)$	Domain i on hierarchy k
$\bar{D}_i^k = (\mathcal{V}_i^k, \mathcal{P}_i^k, \mathcal{L}_i^k)$	Advertised information about domain D_i^k
d_p	Delay of advertised path $p \in \mathcal{P}_i^k$
λ_p^{cap}	Maximum data rate of p shared with paths P_p^λ
$\kappa_p^{\text{cap}} = \kappa_p^{\text{excl}} + \kappa_p^{\text{pool}}$	Compute capacity of p, partly exclusive, partly shared with overlapping paths P_p^κ

TABLE II: Decision Variables

Symbol	Domain	Definition
$x_{c,p}$	$\{0,1\}$	Is an instance of component c is placed on path p?
$\lambda_{c,p}$	$\mathbb{R}_{\geq 0}$	Total ingoing data rate for an instance of c at p
$\mu_{c,p}$	$\mathbb{R}_{\geq 0}$	Total outgoing data rate of an instance of c at p
$\kappa_{c,p}$	$\mathbb{R}_{\geq 0}$	Resource requirements of an instance of c at p
κ_p^{share}	$\mathbb{R}_{\geq 0}$	Path p's share of compute resources taken from resource pool κ_p^{pool} shared with paths in P_p^{comp}
$y_{r,c,c',p,p',p''}^{\text{intra}}$	$\{0,1\}$	Is traffic of request r routed via intra-domain path p'' from an instance of component c at path p to an instance of c' at p'?
$y_{r,c,c',p,p',l}^{\text{inter}}$	$\{0,1\}$	Is traffic of r routed via inter-domain link l from an instance of c at p to an instance of c' at p'?
$\lambda_{r,c,c',p,p',p''}^{\text{intra}}$	$\mathbb{R}_{\geq 0}$	Data rate of r that is routed via intra-domain path p'' from an instance of c at p to an instance of c' at p'
$\lambda_{r,c,c',p,p',l}^{\text{inter}}$	$\mathbb{R}_{\geq 0}$	Data rate of r that is routed via inter-domain link l from an instance of c at p to an instance of c' at p'
$\lambda_{r,c,c',p,p'}^{\text{total}}$	$\mathbb{R}_{\geq 0}$	Total data rate of r from instance c at p to c' at p'
$\lambda_{r,c,c',p}$	$\mathbb{R}_{\geq 0}$	Data rate of r traversing instances of c or c' on p
$\lambda_{r,p}^{\max}$	$\mathbb{R}_{\geq 0}$	Data rate upper bound for all traffic of r on p
$\lambda_p^{\max}$	$\mathbb{R}_{\geq 0}$	Data rate upper bound for all traffic on path p
d_r^{total}	$\mathbb{R}_{\geq 0}$	Total delay for processing and routing request r

A. Objective

$$\min w_1 \cdot \sum_{c \in C, p \in \mathcal{P}_i^k} x_{c,p} + w_2 \cdot \sum_{r \in R} d_r^{\text{total}} \tag{1}$$

The objective in Eq. 1 minimizes the number of placed instances, weighted by w_1, and the total delay for processing all requests, weighted by w_2. This corresponds to lower costs, e.g., for licenses or resources, and better service quality. In our evaluation, we choose a lexicographical order, prioritizing the number of instances over total delay. Other objectives can easily be implemented by choosing suitable weights and possibly including additional decision variables from Table II.

B. Constraints

1) Ingress Traffic and Chaining: Eq. 2 states that the total traffic leaving ingress component c_{in} placed at path p_{in} (at the ingress) to any other instance equals the request's data rate λ_r. Eq. 3 ensures that flow is conserved between chained instances. In particular, all outgoing traffic of c' at p' corresponds to all ingoing traffic modified by function $\mu_{c'}(\lambda)$.

$$\sum_{c' \in C, p \in \mathcal{P}_i^k} \lambda^{\text{total}}_{r,c_{\text{in}},c',p_{\text{in}},p'} = \lambda_r \qquad \forall r \in R \quad (2)$$

$$\mu_{c'}\left(\sum_{c \in C, p \in \mathcal{P}_i^k} \lambda^{\text{total}}_{r,c,c',p,p'}\right) = \sum_{c'' \in C, p'' \in \mathcal{P}_i^k} \lambda^{\text{total}}_{r,c',c'',p',p''}$$
$$\forall r \in R, c' \in C \setminus \{c_{\text{in}}, c_{\text{eg}}\}, p' \in \mathcal{P}_i^k \quad (3)$$

2) Scaling and Placement: Whenever traffic is sent between two instances, Eq. 4 and 5 ensure that both instances indeed exist and are placed accordingly. Using the Big M method, M is a large constant that ensures binary variable $x_{c,p}$ is set to 1 if any traffic traverses the instance. Since we minimize the number of instances in our objective (Eq. 1), the solver sets $x_{c,p} = 0$ if the instance is not traversed by any traffic.

$$\lambda^{\text{total}}_{r,c,c',p,p'} \leq M \cdot x_{c,p} \qquad \forall r \in R, c, c' \in C, p, p' \in \mathcal{P}_i^k \quad (4)$$
$$\lambda^{\text{total}}_{r,c,c',p,p'} \leq M \cdot x_{c',p'} \qquad \forall r \in R, c, c' \in C, p, p' \in \mathcal{P}_i^k \quad (5)$$

Eq. 6 sets variable $\lambda_{c',p'}$ to the total data rate of ingoing traffic for an instance of c' at p'. Eq. 7 and 8 set resource requirements and outgoing data rate for the instance accordingly. Eq. 9 ensures that the resource pool κ_p^{pool} shared between paths in P_p^κ is not over-utilized. Based on this, Eq. 10 guarantees that the overall compute capacity of path p, consisting of partly exclusive and partly shared resources, is not over-utilized by the total resource requirements of instances placed at p.

$$\lambda_{c',p'} = \sum_{r \in R, c \in C, p \in \mathcal{P}_i^k} \lambda^{\text{total}}_{r,c,c',p,p'} \qquad \forall c' \in C, p' \in \mathcal{P}_i^k \quad (6)$$
$$\kappa_{c,p} = \kappa_c(\lambda_{c,p}) \qquad \forall c \in C, p \in \mathcal{P}_i^k \quad (7)$$
$$\mu_{c,p} = \mu_c(\lambda_{c,p}) \qquad \forall c \in C, p \in \mathcal{P}_i^k \quad (8)$$
$$\sum_{p \in P_p^\kappa} \kappa_p^{\text{share}} \leq \kappa_p^{\text{pool}} \qquad \forall p \in \mathcal{P}_i^k \quad (9)$$
$$\sum_{c \in C} \kappa_{c,p} \leq \kappa_p^{\text{excl}} + \kappa_p^{\text{share}} \qquad \forall p \in \mathcal{P}_i^k \quad (10)$$

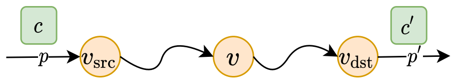


Fig. 4: Flow conservation for routing traffic from c at p with data rate $\lambda^{\text{total}}_{r,c,c',p,p'}$ through node $v \in \mathcal{V}_i^k$ to c' at p'.

3) Routing: Eq. 11 and 12 ensure that the data rate routed via intra-domain paths or inter-domain links does not exceed the total data rate between two instances.

$$\lambda^{\text{intra}}_{r,c,c',p,p',p''} \leq \lambda^{\text{total}}_{r,c,c',p,p'} \qquad \forall r \in R, c, c' \in C, p, p', p'' \in \mathcal{P}_i^k \quad (11)$$
$$\lambda^{\text{inter}}_{r,c,c',p,p',l} \leq \lambda^{\text{total}}_{r,c,c',p,p'} \qquad \forall r \in R, c, c' \in C, p, p' \in \mathcal{P}_i^k, l \in \mathcal{L}_i^k \quad (12)$$

Eq. 13 and 14 set binary routing variables $y^{\text{intra}}_{r,c,c',p,p',p''}$ and $y^{\text{inter}}_{r,c,c',p,p',l}$ using Big M, similar to Eq. 4 and 5.

$$\lambda^{\text{intra}}_{r,c,c',p,p',p''} \leq M \cdot y^{\text{intra}}_{r,c,c',p,p',p''}$$
$$\forall r \in R, c, c' \in C, p, p', p'' \in \mathcal{P}_i^k \quad (13)$$
$$\lambda^{\text{inter}}_{r,c,c',p,p',l} \leq M \cdot y^{\text{inter}}_{r,c,c',p,p',l}$$
$$\forall r \in R, c, c' \in C, p, p' \in \mathcal{P}_i^k, l \in \mathcal{L}_i^k \quad (14)$$

While Eq. 3 ensures flow is conserved between instances of the whole service chain, Eq. 15 ensures flow conservation on intermediate nodes during routing. Particularly, we consider routing via a node v from an instance of component c at path p to an instance of c' at p', where the total traffic has data rate $\lambda^{\text{total}}_{r,c,c',p,p'}$. We denote v_{src} as source node and v_{dst} as destination node as illustrated in Fig. 4.

$$\left(\sum_{p'' \in \mathcal{P}_i^k, p'' \text{ends at } v} \lambda^{\text{intra}}_{r,c,c',p,p',p''} + \sum_{l \in \mathcal{L}_i^k, l \text{ ends at } v} \lambda^{\text{inter}}_{r,c,c',p,p',l}\right)$$
$$- \left(\sum_{p'' \in \mathcal{P}_i^k, p'' \text{starts at } v} \lambda^{\text{intra}}_{r,c,c',p,p',p''} + \sum_{l \in \mathcal{L}_i^k, l \text{ starts at } v} \lambda^{\text{inter}}_{r,c,c',p,p',l}\right)$$
$$= \begin{cases} 0 & \text{if } v_{\text{src}} = v = v_{\text{dst}} \text{ or } v_{\text{src}} \neq v \neq v_{\text{dst}} \\ -\lambda^{\text{total}}_{r,c,c',p,p'} & \text{if } v_{\text{src}} = v \neq v_{\text{dst}} \\ \lambda^{\text{total}}_{r,c,c',p,p'} & \text{if } v_{\text{src}} \neq v = v_{\text{dst}} \end{cases}$$
$$\forall v \in \mathcal{V}_i^k, r \in R, c, c' \in C, p, p' \in \mathcal{P}_i^k \quad (15)$$

4) Link Capacities and Delay: Eq. 16 ensures that the total traffic on inter-domain links does not exceed their capacity.

$$\sum_{r \in R, c, c' \in C, p, p' \in \mathcal{P}_i^k} \lambda^{\text{inter}}_{r,c,c',p,p',l} \leq \lambda_l^{\text{cap}} \qquad \forall l \in \mathcal{L}_i^k \quad (16)$$

The corresponding restriction for intra-domain paths is more complex and thus split into multiple constraints (Eq. 17–20). Eq. 17 ensures that all paths P_p^λ sharing the same bottleneck link as p (including p itself) together do not exceed the maximum data rate of path p. Eq. 18 defines an upper bound for the total data rate on path p based on traffic routed through p (first part) and traffic being processed by instances on p, which may modify the data rate of traversing traffic (second part). The latter data rate can be bounded by considering the maximum data rate on p between any two chained components $c, c' \in C$ placed on p (Eq. 19). In turn, this data rate is calculated in Eq. 20 based on three overlapping parts of traffic: Traffic from an instance of c on another path p' going to c' on p, traffic within p from c, c' instances on p, traffic from c on p to an instance of c' on another path p'. These bounds slightly over-approximate the actual data rate on intra-domain path p, which depends on the refined coordination decisions from child coordinators. However, the key point is that they ensure that link capacities are not exceeded and routing decisions by parent coordinators can be refined into feasible solutions.

$$\sum_{p' \in P_p^{\lambda}} \lambda_{p'}^{\text{max}} \leq \lambda_p^{\text{cap}} \qquad \forall p \in \mathcal{P}_i^k \tag{17}$$

$$\lambda_p^{\text{max}} = \sum_{\substack{r \in R, c, c' \in C, \\ p', p'' \in \mathcal{P}_i^k}} \lambda_{r,c,c',p',p'',p}^{\text{intra}} + \sum_{r \in R} \lambda_{r,p}^{\text{max}} \qquad \forall p \in \mathcal{P}_i^k \tag{18}$$

$$\lambda_{r,p}^{\text{max}} = \max_{c,c' \in C} \lambda_{r,c,c',p} \qquad \forall r \in R, p \in \mathcal{P}_i^k \tag{19}$$

$$\lambda_{r,c,c',p} = \sum_{p \neq p' \in \mathcal{P}_i^k} \lambda_{r,c,c',p',p}^{\text{total}} + \lambda_{r,c,c',p,p}^{\text{total}} + \sum_{p \neq p' \in \mathcal{P}_i^k} \lambda_{r,c,c',p,p'}^{\text{total}}$$
$$\forall r \in R, c, c' \in C, p \in \mathcal{P}_i^k \tag{20}$$

Finally, Eq. 21 calculates the total delay based on all traversed intra-domain paths and inter-domain links, which is minimized in the objective (Eq. 1). It is also possible to bound and minimize the end-to-end delay rather than the total delay using additional variables and constraints. However, we found that it considerably increases complexity and yields similar results as minimizing the total delay and thus focus on the latter.

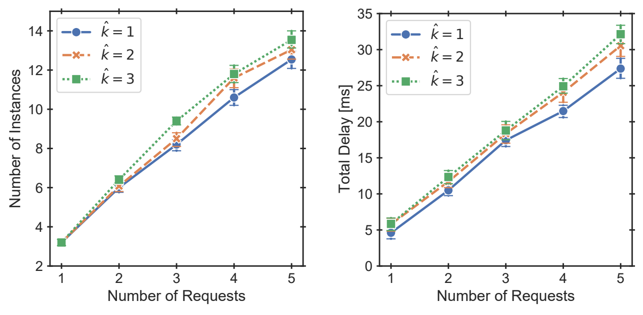
$$d_r^{\text{total}} = \sum_{\substack{c,c' \in C, \\ p,p' \in \mathcal{P}_i^k}} \left(\sum_{p'' \in \mathcal{P}_i^k} y_{r,c,c',p,p',p''}^{\text{intra}} d_{p''} + \sum_{l \in \mathcal{L}_i^k} y_{r,c,c',p,p',l}^{\text{inter}} d_l \right)$$
$$\forall r \in R \tag{21}$$

VI. Evaluation

To evaluate our approach, we implemented it using Python 3.6 and Gurobi 8.11 [39]. Our code is publicly available on GitHub [5] to encourage reproducibility and reuse.

A. Evaluation Setup

We evaluate our approach on real-world network topology Janos [40], which is a US network with 39 nodes and 122 links. We set link delays according to the distance and propagation delay between nodes and chose node and link capacities



(a) Number of instances (b) Total delay

Fig. 5: Comparison of solution quality.

uniformly at random with $\kappa_v^{\text{cap}} \in [8, 64], \lambda_l^{\text{cap}} \in [20, 40]$. Furthermore, we consider two services s_1, s_2 consisting of a load balancer and a deep packet inspector (DPI) with $C_{s_1} = \langle c_{\text{LB}}, c_{\text{DPI}} \rangle$ and $C_{s_2} = \langle c_{\text{DPI}} \rangle$. We generate requests for these services based on a randomly selected subset of 10 ingress and egress nodes and data rate $\lambda_r \in [1, 5]$ GB/s chosen uniformly at random. As evaluation parameter, we increase load by increasing number of requests from 1 to 5.

On these scenarios, we compare the following approaches:

- $\hat{k} = 1$: A typical flat approach with a single centralized coordinator, finding globally optimal solutions (for comparison).
- $\hat{k} = 2$: Our hierarchical approach with two hierarchies: One coordinator on $k = 2$ and two on $k = 1$.
- $\hat{k} = 3$: Our approach with an additional hierarchy: One coordinator on $k = 3$, two on $k = 2$, and four on $k = 1$.

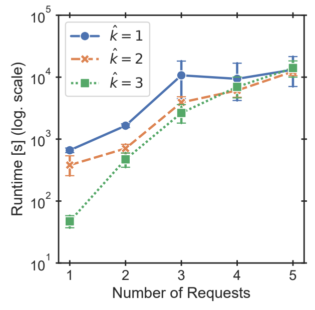
We selected domains within a hierarchy based on node locality (GPS position) using the k-means algorithm [41].

We executed the evaluation on machines with an Intel Xeon E5-2670 CPU, allocating 8 cores at 2.6 GHz and 128 GB RAM per experiment run. The results show the mean and 95 % confidence interval over 25 repetitions.

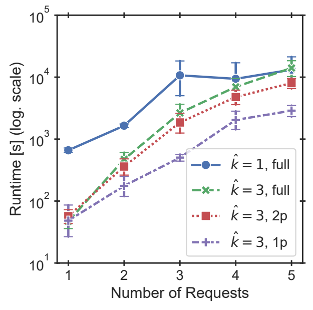
B. Solution Quality

First, we compare the solution quality of our hierarchical approach ($\hat{k} = 2$ and $\hat{k} = 3$) with the centralized approach ($\hat{k} = 1$) in terms of number of placed instances and total delay, which are both minimized in our objective (Sec. V-A). The centralized approach finds globally optimal solutions and thus constitutes a lower bound regarding both metrics.

Fig. 5a shows the number of placed instances with increasing load for the different approaches. Our hierarchical approach finds close-to-optimal solutions with few additional instances compared to the optimal, centralized approach. As expected, the hierarchical approach with $\hat{k} = 2$ is slightly closer to the optimum (3.7 % on avg.) than the one with $\hat{k} = 3$ (8.1 %). This is because the latter has an additional layer of abstraction where more information is aggregated and hidden from the top-level coordinator.

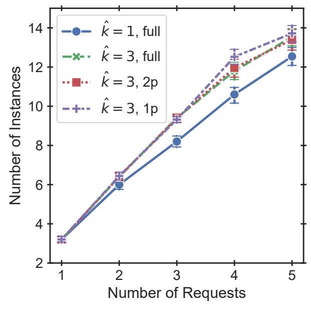


(a) All paths

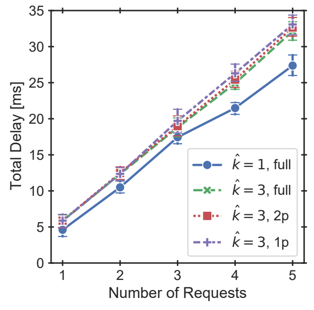


(b) Reduced paths

Fig. 6: Comparison of wall-clock runtime (log. scale).



(a) Number of instances



(b) Total delay

Fig. 7: Solution quality with less advertised information.

Fig. 5b shows similar results for the total delay. To improve comparability, we here set the same fixed number of instances (derived from $\hat{k} = 3$ in Fig. 5a) for all three approaches and only minimize total delay. Hence, $\hat{k} = 1$ now has the same number of instances as $\hat{k} = 2$ and 3 but represents the optimal lower bound regarding total delay. Our hierarchical approach achieves short total delay for both $\hat{k} = 2$ (within 13 % on avg. from optimum) and $\hat{k} = 3$ (17 %).

C. Runtime

While the solution quality of our hierarchical approach is slightly worse than the optimum, its reduced complexity and improved scalability enables much faster execution. Fig. 6a shows the total wall-clock runtimes for each approach on a logarithmic scale when executing all coordinators within one hierarchy in parallel. While numerical optimization with MILPs is generally slow, our hierarchical approach is much faster than the centralized approach. The additional hierarchy of $\hat{k} = 3$ leads to even shorter runtimes than $\hat{k} = 2$ since it hides more complexity from the top-level coordinator. On average, $\hat{k} = 2$ is 88 % and $\hat{k} = 3$ is 470 % faster than $\hat{k} = 1$.

Nevertheless, the runtime grows faster with increasing requests for $\hat{k} = 2$ and $\hat{k} = 3$ than $\hat{k} = 1$. With more requests, more ingress/egress nodes are included in advertised set $\mathcal{V}_i^k$ and, consequently, more intra-domain paths are advertised in $\mathcal{P}_i^k$. Hence, the problem input size and resulting complexity grows in two ways (requests and advertised information) for our hierarchical approach. We further explore the impact of advertised information in Sec. VI-D.

D. Impact of Advertised Information

In Sec. VI-B and VI-C, domains advertise all paths found by solving the maximum flow problem (Sec. IV-B), often multiple paths per source-destination pair. With more nodes $\mathcal{V}_i^k$, this drastically increases the number of advertised paths $\mathcal{P}_i^k$. To improve scalability, we here limit the number of advertised paths per source-destination pair and evaluate the impact on solution quality and runtime. In particular, we compare $\hat{k} = 3$ with all, two, or one advertised paths, denoted as full, 2p, and 1p in figures, respectively.

Fig. 7 shows that, even with fewer advertised paths, $\hat{k} = 3$ still finds close-to-optimal solutions. Fewer advertised paths lead to slightly lower solution quality, i.e., more instances (1.4 % on avg. with one vs. all paths) or higher delay (2.8 %), since coordinators lack some information. At the same time, Fig. 6b shows that advertising fewer paths considerably reduces complexity and improves runtime. With just one advertised path per source-destination pair, $\hat{k} = 3$ is on average 3.4x faster than with full advertised paths and 10.7x faster than $\hat{k} = 1$. Moreover, runtime grows slower with increasing requests when advertising fewer paths.

VII. Conclusion

Our approach for hierarchical network and service coordination combines the benefits of centralized and distributed approaches. It achieves close-to-optimal solution quality at a fraction of the runtime compared to optimal, centralized solutions. To control the trade-off between optimal solution quality and fast runtime, operators can adjust the number of hierarchical levels and the amount of information advertised from lower to higher levels. The resulting MILPs can be solved in parallel for all coordinators at one level and can easily be adjusted to optimize any objective of interest. We believe our proposed hierarchical approach using bottom-up information advertisement and top-down coordination to be a useful framework in general. In future work, this framework may be applied to other kinds of optimization methods such as heuristics to improve their performance and scalability.

Acknowledgments

This work has received funding from the German Research Foundation (DFG) within the Collaborative Research Centre "On-The-Fly Computing" (SFB 901) and the German Federal Ministry of Education and Research through Software Campus grant 01IS17046 (RealVNF).

References

[1] W. Li, Y. Lemieux, J. Gao, Z. Zhao, and Y. Han, "Service mesh: Challenges, state of the art, and future research opportunities," in *IEEE International Conference on Service-Oriented System Engineering (SOSE)*. IEEE, 2019, pp. 122–1225.

[2] R. Mijumbi, J. Serrat, J.-L. Gorricho, N. Bouten, F. De Turck, and R. Boutaba, "Network function virtualization: State-of-the-art and research challenges," *IEEE Communications surveys & tutorials*, vol. 18, no. 1, pp. 236–262, 2015.

[3] S. Dräxler, H. Karl, and Z. Á. Mann, "Jasper: Joint optimization of scaling, placement, and routing of virtual network services," *IEEE Transactions on Network and Service Management*, vol. 15, no. 3, pp. 946–960, 2018.

[4] N. Kang, Z. Liu, J. Rexford, and D. Walker, "Optimizing the "one big switch" abstraction in software-defined networks," in *ACM Conference on Emerging Networking Experiments and Technologies (CoNEXT)*, 2013, pp. 13–24.

[5] M. Jürgens and S. Schneider, "Hierarchical Network and Service Coordination GitHub Repository," https://github.com/CN-UPB/hierarchical-coordination, 2020.

[6] J. G. Herrera and J. F. Botero, "Resource allocation in NFV: A comprehensive survey," *IEEE Transactions on Network and Service Management*, vol. 13, no. 3, pp. 518–532, 2016.

[7] C.-H. Hong and B. Varghese, "Resource management in fog/edge computing: A survey on architectures, infrastructure, and algorithms," *ACM Computing Surveys (CSUR)*, vol. 52, no. 5, pp. 1–37, 2019.

[8] Z. Á. Mann, "Allocation of virtual machines in cloud data centers—a survey of problem models and optimization algorithms," *ACM Computing Surveys (CSUR)*, vol. 48, no. 1, pp. 1–34, 2015.

[9] H. Moens and F. De Turck, "VNF-P: A model for efficient placement of virtualized network functions," in *IEEE International Conference on Network and Service Management*, 2014, pp. 418–423.

[10] F. Bari, S. R. Chowdhury, R. Ahmed, R. Boutaba, and O. C. M. B. Duarte, "Orchestrating virtualized network functions," *IEEE Transactions on Network and Service Management*, vol. 13, no. 4, pp. 725–739, 2016.

[11] D. Bhamare, M. Samaka, A. Erbad, R. Jain, L. Gupta, and H. A. Chan, "Optimal virtual network function placement in multi-cloud service function chaining architecture," *Computer Communications*, vol. 102, pp. 1–16, 2017.

[12] S. Ahvar, H. P. Phyu, S. M. Buddhacharya, E. Ahvar, N. Crespi, and R. Glitho, "CCVP: Cost-efficient centrality-based VNF placement and chaining algorithm for network service provisioning," in *IEEE Conference on Network Softwarization (NetSoft)*. IEEE, 2017, pp. 1–9.

[13] T.-W. Kuo, B.-H. Liou, K. C.-J. Lin, and M.-J. Tsai, "Deploying chains of virtual network functions: On the relation between link and server usage," *IEEE/ACM Transactions on Networking*, vol. 26, no. 4, pp. 1562–1576, 2018.

[14] S. Dräxler, S. Schneider, and H. Karl, "Scaling and placing bidirectional services with stateful virtual and physical network functions," in *IEEE Conference on Network Softwarization (NetSoft)*, 2018.

[15] M. C. Luizelli, W. L. da Costa Cordeiro, L. S. Buriol, and L. P. Gaspary, "A fix-and-optimize approach for efficient and large scale virtual network function placement and chaining," *Computer Communications*, vol. 102, pp. 67–77, 2017.

[16] S. Schneider, R. Khalili, A. Manzoor, H. Qarawlus, R. Schellenberg, H. Karl, and A. Hecker, "Self-driving network and service coordination using deep reinforcement learning," in *IEEE/IFIP International Conference on Network and Service Management (CNSM)*. IEEE/IFIP, 2020.

[17] S. Schneider, L. D. Klenner, and H. Karl, "Every node for itself: Fully distributed service coordination," in *IEEE/IFIP International Conference on Network and Service Management (CNSM)*. IEEE/IFIP, 2020.

[18] I. Houidi, W. Louati, and D. Zeghlache, "A distributed virtual network mapping algorithm," in *IEEE International Conference on Communications*. IEEE, 2008, pp. 5634–5640.

[19] W. Findeisen, F. N. Bailey, M. Brdys, K. Malinowski, P. Tatjewski, and A. Wozniak, *Control and coordination in hierarchical systems*. John Wiley & Sons, 1980.

[20] M. D. Mesarovic, D. Macko, and Y. Takahara, *Theory of Hierarchical, Multilevel, Systems, Volume 68*. Elsevier Science, 2000.

[21] A. Fischer, J. F. Botero, M. T. Beck, H. De Meer, and X. Hesselbach, "Virtual network embedding: A survey," *IEEE Communications Surveys & Tutorials*, vol. 15, no. 4, pp. 1888–1906, 2013.

[22] F. Samuel, M. Chowdhury, and R. Boutaba, "PolyViNE: Policy-based virtual network embedding across multiple domains," *Journal of Internet Services and Applications*, vol. 4, no. 1, p. 6, 2013.

[23] T. Ghazar and N. Samaan, "Hierarchical approach for efficient virtual network embedding based on exact subgraph matching," in *IEEE Global Telecommunications Conference (GLOBECOM)*. IEEE, 2011, pp. 1–6.

[24] D. King and A. Farrel, "The application of the path computation element architecture to the determination of a sequence of domains in MPLS and GMPLS," IETF, Internet Request for Comments RFC 6805, 2012.

[25] D. Dhody, Y. Lee, D. Ceccarelli, J. Shin, and D. King, "Hierarchical stateful path computation element (PCE)," IETF, Internet Request for Comments RFC 8751, 2020.

[26] R. Douville, J.-l. Rougier, S. Secci *et al.*, "A service plane over the PCE architecture for automatic multidomain connection-oriented services," *IEEE Communications Magazine*, vol. 46, no. 6, pp. 94–102, 2008.

[27] M. Chamania and A. Jukan, "A survey of inter-domain peering and provisioning solutions for the next generation optical networks," *IEEE Communications Surveys & Tutorials*, vol. 11, no. 1, pp. 33–51, 2009.

[28] S. Secci, J.-L. Rougier, and A. Pattavina, "On the selection of optimal diverse AS-paths for inter-domain IP/(G) MPLS tunnel provisioning," in *Telecommunication Networking Workshop on QoS in Multiservice IP Networks*. IEEE, 2008, pp. 235–241.

[29] J. M. Kleinberg, "Single-source unsplittable flow," in *IEEE Conference on Foundations of Computer Science*. IEEE, 1996, pp. 68–77.

[30] W. Ma, O. Sandoval, J. Beltran, D. Pan, and N. Pissinou, "Traffic aware placement of interdependent NFV middleboxes," in *IEEE International Conference on Computer Communications (INFOCOM)*. IEEE, 2017, pp. 1–9.

[31] M. Peuster, S. Schneider, and H. Karl, "The softwarised network data zoo," in *IEEE/IFIP International Conference on Network and Service Management (CNSM)*. IEEE/IFIP, 2019.

[32] S. B. Schneider, N. P. Satheeschandran, M. Peuster, and H. Karl, "Machine learning for dynamic resource allocation in network function virtualization," in *IEEE Conference on Network Softwarization (NetSoft)*. IEEE, 2020.

[33] S. Dräxler and H. Karl, "SPRING: Scaling, placement, and routing of heterogeneous services with flexible structures," in *IEEE Conference on Network Softwarization (NetSoft)*. IEEE, 2019, pp. 115–123.

[34] L. Popa, G. Kumar, M. Chowdhury, A. Krishnamurthy, S. Ratnasamy, and I. Stoica, "FairCloud: Sharing the network in cloud computing," in *ACM SIGCOMM Conference*, 2012, pp. 187–198.

[35] S. Schneider, S. Dräxler, and H. Karl, "Trade-offs in dynamic resource allocation in network function virtualization," in *IEEE GLOBECOM Workshops*. IEEE, 2018, pp. 1–3.

[36] Cloud Native Computing Foundation, "Kubernetes: Production-grade container orchestration," https://kubernetes.io/ (accessed Jan 31, 2020), 2020.

[37] L. R. Ford and D. R. Fulkerson, "Maximal flow through a network," in *Classic Papers in Combinatorics*. Springer, 2009, pp. 243–248.

[38] J. Edmonds and R. M. Karp, "Theoretical improvements in algorithmic efficiency for network flow problems," *Journal of the ACM (JACM)*, vol. 19, no. 2, pp. 248–264, 1972.

[39] G. Optimization, "Gurobi solver," https://www.gurobi.com/, 2020.

[40] S. Orlowski, M. Pióro, A. Tomaszewski, and R. Wessäly, "SNDlib 1.0–Survivable Network Design Library," in *International Network Optimization Conference (INOC)*, 2007. [Online]. Available: sndlib.zib.de

[41] A. K. Jain, "Data clustering: 50 years beyond K-means," *Pattern recognition letters*, vol. 31, no. 8, pp. 651–666, 2010.