Use What You Know: Network and Service Coordination Beyond Certainty

Stefan Werner
Paderborn University, Germany
stwerner@mail.upb.de

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

Holger Karl
Hasso Plattner Institute,
University of Potsdam, Germany
holger.karl@hpi.de

***Abstract*—Modern services often comprise several components, such as chained virtual network functions, microservices, or machine learning functions. Providing such services requires to decide how often to instantiate each component, where to place these instances in the network, how to chain them and route traffic through them. To overcome limitations of conventional, hard-wired heuristics, deep reinforcement learning (DRL) approaches for self-learning network and service management have emerged recently. These model-free DRL approaches are more flexible but typically learn tabula rasa, i.e., disregard existing understanding of networks, services, and their coordination. Instead, we propose FutureCoord, a novel model-based AI approach that leverages existing understanding of networks and services for more efficient and effective coordination without time-intensive training. FutureCoord combines Monte Carlo Tree Search with a stochastic traffic model. This allows FutureCoord to estimate the impact of future incoming traffic and effectively optimize long-term effects, taking fluctuating demand and Quality of Service (QoS) requirements into account. Our extensive evaluation based on real-world network topologies, services, and traffic traces indicates that FutureCoord clearly outperforms state-of-the-art model-free and model-based approaches with up to 51% higher flow success ratios.**

***Index Terms*—network management, service management, AI, Monte Carlo Tree Search, model-based, QoS**

I. Introduction

Services consisting of multiple chained components trend towards softwarization and virtualization, e.g., chained virtual network functions (VNFs) [1], microservices in a service mesh [2], or machine learning functions [3]. The gained flexibility comes with novel challenges regarding network and service coordination: The service components need to be scaled and instantiated across different network nodes. Incoming traffic needs to be assigned to and balanced between the placed instances. In doing so, coordination needs to take limited node/link capacities and Quality of Service (QoS) requirements into account and adapt to constant changes in demand.

Existing work conventionally addresses network and service coordination using hand-crafted heuristics designed by experts. Such heuristics are typically tailored to specific scenarios and explicitly define *how* to coordinate network and services through rigid algorithms and rules. This is problematic when operational reality diverges from the assumed scenario, again requiring time-consuming manual adjustments by experts.

To overcome hard-wired heuristics, recent work proposes self-learning network and service coordination, mostly using model-free deep reinforcement learning (DRL). These approaches require training on a given network environment, typically starting from scratch, i.e., with completely random behavior. Training can be extremely time- and resource-intensive. For example, OpenAI Five, a DRL approach mastering the Dota 2 video game, was trained on more than 100,000 CPU cores for 10 months [4]. Moveover, DRL training is not guaranteed to converge [5], [6] and outcomes can drastically differ depending on the random training seed [7]. A core problem of model-free DRL is that these approaches start with no knowledge or understanding of the problem or environment at hand. Consequently, they are very flexible but also must learn everything by themselves through trial and error.

In particular, DRL approaches for network and service coordination are ignorant of the existing understanding of networks, services, traffic and their interplay. E.g., service components are typically known and their resource requirements can be profiled [8], [9]. Similarly, the topology and available capacities in a given network are typically rather static and well known. Analysis of recorded traffic traces can help to understand typical traffic patterns; recent advances in traffic forecasting even allow predicting upcoming traffic [10]–[13] and QoS requirements [14] with reasonable certainty. Neglecting all this information, current model-free DRL approaches must learn everything implicitly by themselves, which is not only time-consuming and resource-expensive but also known to be notoriously hard for real-world problems (e.g., due to non-stationary traffic changes) [15].

We propose instead a novel model-based AI approach, called FutureCoord, that leverages a model of the network to effectively coordinate network and services. Similar to AlphaZero [16], our approach uses Monte Carlo Tree Search (MCTS) [17], where a model defines the rules of the game but not *how* to win. MCTS uses the model to find a winning strategy itself. Similarly, our model defines relevant, well-understood characteristics in networks but does not dictate *how* to coordinate the network and services. It is also simple to extend the model, e.g., including new research findings, without redesigning the coordination approach itself.

A key novelty of our FutureCoord approach, going beyond related model-based approaches [18]–[20], is the combination of MCTS with a stochastic traffic model. The traffic model forecasts upcoming traffic flows, e.g., based on learning models from traffic traces recorded over the recent past. Rather

than coordinating each incoming flow in isolation, this allows us to consider likely future flows explicitly and anticipate their long-term effects. For example, FutureCoord places service instances at strategic locations in the network where they can be reused by other flows, avoiding starting new instances and thus minimizing resource requirements. Similarly, it avoids blocking resources that are critical for upcoming QoS-sensitive or resource-heavy flows, ultimately leading to higher flow success rates. Our trace-driven evaluation based on real-world data indicates that FutureCoord outperforms a related model-based approach, a state-of-the-art model-free DRL agent and a heuristic by up to 51% more successful flows. In doing so, FutureCoord is robust to changes in load, network topology, traffic pattern and still obtains good results with inaccurate forecasts. Overall, our contributions are:

- We propose FutureCoord, a novel model-based AI approach, combining MCTS with a stochastic traffic model to optimize network and service coordination for upcoming demands and QoS requirements.
- Our approach adapts to varying demands, QoS requirements and heterogeneous services, outperforming existing model-free and model-based approaches by up to 51%.
- We open-source our code [21] to encourage reproducibility and reuse of our work.

II. Related Work

Most existing work on service coordination uses conventional approaches and formulates mixed-integer linear programs (MILP) or hand-crafts heuristics [22]. The design of heuristics becomes more challenging if all aspects including scaling, placing, assigning and routing are considered. Related work thus mostly focuses on aspects in isolation. For example, authors optimize the energy consumption [23] or setup costs [24] of placements but do not consider their assignment to flows. Still, other authors [25], [26] tackle the full service coordination problem. For example, GRC-VNE [26] balances server loads using node rankings and places instances predominantly where plenty resource capacity remains. Like ours, recent work on online heuristics [27]–[30] uses predictions of future demands. Besides drawbacks of predefined rule sets, the main differences are that *a*) they use forecasts to provision services proactively; we plan long-term effects of decisions. Both aspects could complement each other in future work. *b*) They predict requested flow data rates or components for upcoming time intervals. Our traffic model also accounts for QoS requirements including the max. delay bound of flows and allows quantifying the uncertainty of their estimates.

Recently, DRL has emerged as a promising paradigm to address the time-varying nature of traffic [31]. One major advantage is that coordination policies are self-learned from experience and do not rely on model abstractions of the network [32]–[34]. Still, many authors settle for a compromise and integrate simulation models to assist their learned coordination policy. The model is then used to simulate and test whether decisions constitute feasible placements before carrying them out in practice [35]–[37]. Others use heuristics to identify what placements are viable in the model with priority to decisions favored by the DRL policy [38] or guide their exploration during training towards suitable solutions to accelerate learning [39], [40]. We believe that such work suffers from requiring models without taking full advantage of them like the ability to plan ahead (which we pursue).

It is often claimed that DRL naturally adjusts to time-varying traffic [35]. Meanwhile, the data passed to the policy often lacks information (e.g. time of day) that would allow characteristics of the traffic to be learned implicitly [33]–[35]. So far, related work mostly avoids this problem and, e.g., trains separate DRL policies for each considered time interval [36]. A promising direction of research is thus to use traffic forecasts to assist DRL policies. For example, Qu et al. [13] include predictions of demands and points in time when traffic changes into their agent's state. We view our algorithm as a step towards this direction and believe that an integration of its forecast- and model-based search with DRL à la AlphaZero seems promising for future work. Despite their DRL focus, the work by Pei et al. [36] is closely related to ours. Like us, they use a model and forecasts. Still, major differences are that *a*) their model is used to test the feasibility of decisions, ours to strategically plan service coordination *b*) they assume exact forecasts over the upcoming time interval, we do not.

Existing work already uses MCTS for service coordination [18], [19], [41]. Most related to ours is MaVEn-S [19] that grows search trees of decisions: *a*) Like us, it simulates the placement of service components and their connection with routes of lowest delay at each step. *b*) Unlike us, it minimizes operator costs; we maximize flow completion rates. In online decision making, this poses a significantly tougher challenge in the long-run. In fact, any successful candidate deployment seems equally viable. The search must therefore decide between seemingly equivalent options and, e.g., resort to carefully designed evaluation heuristics to evaluate what and where resources are used. Still, even short-term optimal resource usage per flow not necessarily coincides with long-term maximal flow completion rates. *c*) Their admission control policy accepts any flow more profitable than an empirical threshold. Ours adjusts dynamically to varying circumstances; it does not need any pre-defined threshold. *d*) they plan deployments until flows are either completed or dismissed. What operator costs arise for candidate deployments is calculated per flow. Potential loss of revenue for future demands due to long-term effects of decisions is not accounted for. Our work evaluates deployments by what completion rates are achievable long-term using forecasts; it does not greedily maximize the current flow's benefit at the expense of future demands.

III. Problem Statement

We coordinate services, including the deployment of instances, the admission control of flows, their assignment to instances and routing through the network. This section formulates the problem, decision variables and the objective function.

A. Problem Parameters

Formally, the substrate network $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is an undirected graph of nodes $v \in \mathcal{V}$ and bidirectional links $\{u, v\} \in \mathcal{E}$. Each node $v \in \mathcal{V}$ provides compute capacity cpu_v and memory capacity mem_v. Bidirectional links $\{u, v\} \in \mathcal{E}$ govern the amount of traffic that may transit among u and v in either direction by (shared) data rate $\lambda_{\{u,v\}}$. Routing traffic across link $\{u, v\}$ imposes delay $\ell_{\{u,v\}}$. For simplicity, we define $\ell_{\{u,v\}}$ as the propagation delay (ignoring queuing delay).

Flows $f = (o_f, v_f^{\text{ingr}}, v_f^{\text{egr}}, t_f, \delta_f, \lambda_f, \pi_f) \in \mathcal{F}$ enter the network starting at time t_f at their ingress v_f^{ingr}. They arrive according to sequence $Z = (f_1, \ldots, f_N)$. When flow f is admitted, traffic of data rate λ_f enters at ingress v_f^{ingr}, is steered through the network and finally departs at egress v_f^{egr}. Any node can be an ingress, egress or serve as both endpoints. The delay imposed on flows f must not violate their flow-specific maximum end-to-end delay bound π_f. For duration δ_f, flows f request services as linear chains $o_f = \left(c_1, \ldots, c_{m_{o_f}}\right) \in \mathcal{O}$ composed of components $c \in \mathcal{C}$ such as firewalls or web proxies; duration δ_f is unknown to the coordinator. Here, $\mathcal{C}$ denotes the set of components offered by the operator. Along the route established from its ingress to egress, any flow f must traverse instances of all components processing it in the order of o_f to successfully complete the service. Instances consume compute and memory capacities dependent on the amount of traffic they process [8], [9]. For example, instances typically require large amounts of memory once upon their deployment (overhead of VM) and grow slower when instances are scaled up [42]. Our work considers inter-node service coordination including the scaling and placement of instances along with the assignment and admission of flows. Within single nodes, we assume that systems such as Kubernetes [43] automatically scale instances up and down to accommodate varying loads (intra-node scaling). To serve a total data rate λ of possibly many flows, instances of component c consume compute capacity $h_c^{\text{cpu}}(\lambda)$ and memory capacity $h_c^{\text{mem}}(\lambda)$. Both define the resource consumption of instances possibly non-linear in load; they are assumed known to the operator. The amount of compute and memory capacities instances consume to process a given data rate can, in practice, be learned component-wise via supervised learning and benchmarking [44]. Finally, successful flows f depart once their duration passes at time $t_f + \delta_f$. Any instance serving f is then automatically scaled down or terminated in case it processes no more active flows.

B. Decision Variables & Network State

1) Decision Variables: We consider embedding services $o \in \mathcal{O}$ to a network (placement & assignment), routing flows $f \in Z$ among instances thereof (routing) after deciding their admission (admission control). For each flow arriving over time T according to sequence Z, let binary decision variable $x_f \in \{0, 1\}$ denote whether f is admitted upon entering the network or rejected and immediately dropped otherwise. Any successful flow f must traverse instances of all components specified by its service o_f in-order. The i-th component $o_{f,i}$ of flow f is then served by the node with decision variable $y_{f,i} \in V$. For each successful flow f, routes that steer traffic from its ingress v_f^{ingr} across assigned instances and finally to egress node v_f^{egr} must be established. We introduce decision variable $p_{f,i}$ to denote the path forwarding traffic from placements at $y_{f,i}$ to $y_{f,i+1}$ for succeeding components $o_{f,i}$ and $o_{f,i+1}$. Note that $p_{f,0}$ and $p_{f,m_{o_f}}$ establish routes from the ingress and towards the egress node, respectively. This work considers routes along paths of shortest delay.

2) Placement, Assignment & Routing Constraints: For resource consumption of instances non-linear in data rate, the decrease of capacity at node $y_{f,i}$ after being assigned flow f depends not just on the flow's data rate λ_f but also on the instance's prior load. In fact, an instance of component c at node v processes the aggregated data rate $\lambda_{v,c}(t) = \sum_{f \in A(t)} \sum_{o_{f,i}=c} \lambda_f \cdot \mathbb{1}_{\{v=y_{f,i}\}}$ where $A(t)$ denotes the set of flows active at time t and $\mathbb{1}_{\{v=y_{f,i}\}}$ indicates whether node v serves the i-th component $o_{f,i}$ of flow f. We require that the utilized compute capacity $\sum_{c \in \mathcal{C}} h_c^{\text{cpu}}(\lambda_{v,c}(t))$ and allocated memory capacity $\sum_{c \in \mathcal{C}} h_c^{\text{mem}}(\lambda_{v,c}(t))$ do not exceed compute cpu_v and memory mem_v capacities of node v. To avoid oversubscribing links, routes must steer traffic across links $\{u, v\}$ with capacity $\lambda_{\{u,v\}}$ at least the total data rate $\sum_{f \in A(t)} \sum_i \lambda_f \cdot \mathbb{1}_{\{\{u,v\} \in p_{f,i}\}}$ of flows traversing it. The end-to-end delay $\sum_i \sum_{\{u,v\} \in p_{f,i}} \ell_{\{u,v\}}$ of flow f routed along paths $p_{f,i}$ must comply with its maximum delay bound π_f.

C. Objective Function

We assign equal priority to flows and maximize the rate of successfully completed flows $F_{\text{succ.}}$ over time period T:

$$\lim_{T \to \infty} \frac{1}{T} F_{\text{succ.}} \tag{1}$$

Effective admission control must dynamically decide whether flow demands can and should be fulfilled, or if their deployment degrades completion rates in the long-run. For example, it should drop flows that cannot be assigned to already existing instances (avoids memory overhead of further deployments) in favor of possible future flow arrivals once capacities become scarce. To encourage an efficient usage of resources by virtue of sharing, instances should be placed at strategic locations. Flows with tight max. delay bounds make it necessary to preserve capacities along shortest path routes.

IV. FutureCoord: Model-Based AI

Our work overcomes limitations of other approaches that maximize the utility of deployments per flow at the expense of future demands. To do so, we integrate traffic models into the planning stage and evaluate decisions by their long-term effects which enables more extensive and careful planning when appropriate. We thus trade-off the computational budget available for planning and the time over which future demands are considered. In practice, the investment of further effort may be particularly worthwhile for flows of high resource demands or long duration times (if known). Otherwise, it can simplify along the lines of MaVEn-S [19] and coordinate services without explicitly considering future demands.

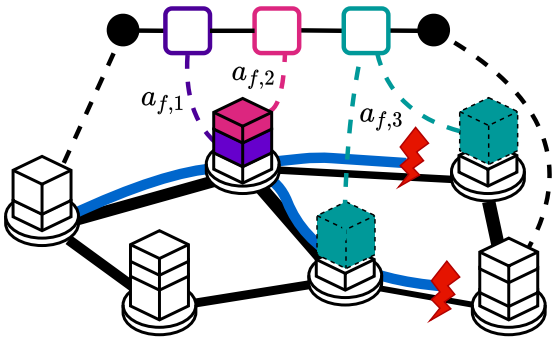


Fig. 1: Components are iteratively assigned to instances. Any of the service's last candidate placements (choices for $a_{f,3}$) are invalid; no route towards the flow's egress can be established.

A. Input-Driven Markov Decision Process

We formalize the online coordination of services as an input-driven Markov Decision Process (ID-MDP) [45] that extends the standard MDP formulation by stochastic process $\mathcal{P}_Z$. Process $\mathcal{P}_Z$ governs when which flows enter the network, i.e., defines the distribution of flow interarrival and duration times, flow data rates, max. delay bounds as well as what services they request. The sequence Z of flows entering over time T follows process $\mathcal{P}_Z$. We assume that flow arrivals are independent of whether prior flows were completed. To this end, the ID-MDP is given by $(\mathcal{S}, \mathcal{A}, \mathcal{Z}, \mathcal{P}_S, \mathcal{P}_Z, \mathcal{R})$ where set $\mathcal{S}$ denotes possible network model states (cmp. Sec. III), action set $\mathcal{A}$ defines coordination decisions, set $\mathcal{Z}$ outlines possible flow arrivals Z over time T and $\mathcal{R}$ is the reward function. Model-based approaches, like ours, use the environment dynamics $\mathcal{P}_S$ (defined in Sec. III) for planning ahead and assume them to be known. Sophisticated simulations of the, typically unknown, real-world dynamics can be used to further close the gap between model assumptions and coordination in practice (e.g. [46]). Below, we define $\mathcal{A}$ and $\mathcal{R}$ in detail.

1) Action Space $\mathcal{A}$: We decide whether to admit flows f and, if so, how their services o_f are deployed. Proceeding iteratively through the chain of components $c_1, \ldots, c_{m_{o_f}}$, each action $a_{f,i} \in \mathcal{V}$ decides which node processes flow f by what instance. For the first component, there is an additional control action ($a_{f,1} = 0$) that dismisses the entire flow. Otherwise, actions either scale up an existing instance on node $a_{f,i}$ to serve the increased load or start up a new instance of component $o_{f,i}$. For example, Fig. 1 shows an assignment of the first and second component (purple and red) by actions $a_{f,1}$ and $a_{f,2}$ to the same node. At each assignment, we interconnect placement location $a_{f,i}$ with the prior instance at $a_{f,i-1}$ or the flow's ingress at v_f^{ingr}, using a shortest-delay path of links with enough capacity. Route $p_{f,m_{o_f}}$ towards flow f's egress is then set up after assigning its final component with action $a_{f,m_{o_f}}$. Based on our model, we adjust the action set $\mathcal{A}$ to what coordination decisions are valid (cmp. Sec. III) at each step. For example, Fig. 1 shows that regardless of where the third component (green) is placed by action $a_{f,3}$, no route (blue) towards the flow's egress can be set up, since its data rate (width of route) exceeds residual link capacities (width of links). If no valid candidate remains for the next component's assignment, the flow cannot be completed and is dropped. All instances serving it (purple and red) are either scaled down or terminated. Then, we await the next flow's arrival.

2) Reward $\mathcal{R}$: To maximize the ratio of successful flows, we attribute rewards of $+1$ upon their completion and zero for any other assignment or control action. This maximizes our objective (Eq. 1) directly. Typically, sparse binary feedback signals, like ours, are not well suited for online planning tasks. In fact, any candidate deployment that completes the current flow is attributed equal return. We thus lack the insight what decisions allow high completion rates of future flows. Existing work usually introduces auxiliary functions such as resource usage [18], [19] to evaluate candidate decisions. However, these auxiliaries are often ill-equipped, since maximum utility in terms of the auxiliary not necessarily coincides with the objective (flow completions). Instead, we optimize service coordination directly by the sparse binary feedback and do not resort to any proxy thereof, as later discussed in Sec. IV-B2.

B. Future Flow Sampling-Based MCTS

We coordinate services using the well-known MCTS algorithm and refer readers unfamiliar with its procedure to Appx. A. In short, MCTS iteratively grows search trees towards regions where high accumulated rewards (returns) were achieved beforehand. It estimates what return is expected when coordination decisions $a \in \mathcal{A}$ are selected in network model state $s \in \mathcal{S}$ as action values $Q(s,a) = \mathbb{E}\big[r + \gamma V(s') \mid s, a\big]$. This includes the immediate reward r for action a and undiscounted ($\gamma = 1$) state value $V(s')$ of successor model state s'.

Existing work [18]–[20] on model-based service coordination terminates its search once the current flow is either completed or dismissed. Action values $Q(s,a)$ then estimate operator costs or energy consumption for the current flow and its deployment, but do not include those of future flows. In the long-run, this shallow search optimizes flow utilities often at the expense of future demands (cmp. Sec. III-C). To overcome this limitation, we propose to plan beyond the current flow's coordination and consider forecasts. Candidate deployments are then not evaluated by auxiliaries but in terms of long-term completion rates for forecast flows. This section outlines the traffic model (Sec. IV-B1), its integration into the search (Sec. IV-B2) and outlines the algorithm in-depth (Sec. IV-B3).

1) Flow Forecasts: To consider long-term effects of decisions, we use traffic models that estimate the possibly non-stationary distributions of future flow arrival and duration times, their data rates, max. delay bounds and services. In practice, such models can be learned from previously recorded traces, e.g., using Gaussian Process Regression [10], [13], [14]. Sec. V-C shows that estimated distributions need not be exact for this scheme to succeed. In fact, our approach is robust to over- and underestimations of flow properties as well as faulty flow arrival patterns.

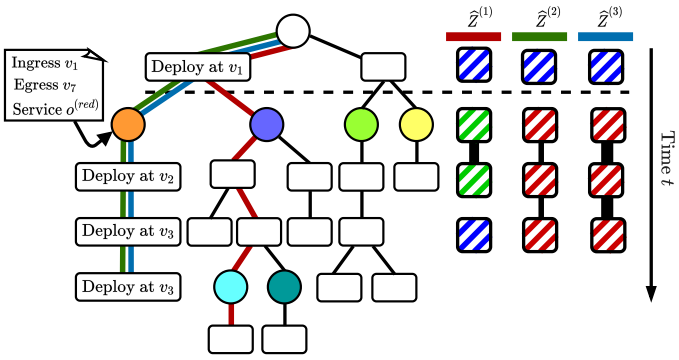


Fig. 2: FutureCoord plans service coordination beyond the current flow (left) with forecasts of future demands (right).

2) Sampling-Based Planning: Each iteration of the search first samples a forecast $\widehat{Z}$ and then performs the selection, expansion, rollout and backup phases based on what flows it predicts. Any decision that makes inefficient use of capacities or congests critical routes likely causes many forecast flows to be dropped afterwards. Rollouts then assign low return (number succ. flows) when decisions complete flows at the expense of forecast flows; the binary feedback becomes informative and favors decisions with high long-term completion rates.

Fig. 2 shows how we unify planning with various forecasts. Iterations follow the same search path if forecast flows *a*) enter and depart at identical ingresses and egresses *b*) specify the same service *c*) are assigned to the same nodes. Then, tree nodes average returns only for related iterations, i.e., simulations of flows at similar positions (ingress & egress) with the same service, but possibly different data rates, max. delay bounds, arrival and duration times. Upon flows entering the network, we insert delimiter nodes into the search tree to distinguish unrelated iterations. For example, the simulation for $\widehat{Z}^{(2)}$ and $\widehat{Z}^{(3)}$ (green & blue search path) assign the current flow's single component (blue stripes; above dotted line) to node v_1. Both predict that flows for the three-component service (red stripes) follow, enter at ingress v_1 and depart at v_7. We thus insert a delimiter node (orange) unique to their ingress, egress and service. Both iterations then continue to simulate the assignment to nodes v_2, v_3, v_3, receiving rewards for completed flows along the way. In this case, the flow forecast by $\widehat{Z}^{(3)}$ has higher data rate (width of links between components) and, for this reason, cannot establish any egress route, receiving zero reward. Assuming that the simulation for $\widehat{Z}^{(2)}$ completes its forecast flow, the last visited tree node averages returns $+1$, $+0$ and is assigned an action value $Q(s,a)$ of $1/2$. If both iterations complete the current flow, the root's left child averages returns $+2$, $+1$ along with results from prior iterations.

3) Algorithm: Alg. 1 outlines our proposed search procedure. First, the selection phase (ln. 6-8) traverses the search tree until a leaf node is encountered. In each step, it selects among successors, simulates the chosen action and stores its reward. Second, we randomly select and simulate an unvisited

Algorithm 1: FutureCoord

1 $\mathcal{T} \leftarrow$ initialize search tree
2 **while** *Computational Budget* **do**
3 $\quad \widehat{Z} \leftarrow$ sample α_{flows} forecast flows
4 $\quad (s,a) \leftarrow (s_{\text{root}}, \bullet)$
5 $\quad R[\,] \leftarrow \emptyset$ ▷ stores rewards
6 $\quad$ **while** *children* (s',a') *of* (s,a) *all visited* **do**
7 $\quad\quad (s,a) \leftarrow \underset{(s',a')}{\arg\max}\, Q(s',a') + 2C\sqrt{\frac{2\ln N(s,a)}{N(s',a')}}$
8 $\quad\quad R[s,a] \leftarrow$ simulate action a; receive reward r
9 $\quad (s,a) \leftarrow$ select random unvisited child
10 $\quad$ **if** (s,a) *is initial action of forecast flow* f **then**
11 $\quad\quad$ insert delimiter node; assign zero reward to it
12 $\quad R[s,a] \leftarrow$ simulate action a; receive reward r
13 $\quad$ expand $\mathcal{T}$ by child tree node (s,a)
14 $\quad R \leftarrow$ MC return R from simulation with future flow arrivals $\widehat{Z}$
15 $\quad$ **for** *visited tree node* (s,a) *until root* **do**
16 $\quad\quad Q(s,a) \leftarrow Q(s,a) + \frac{r - Q(s,a)}{N(s,a)+1}$
17 $\quad\quad N(s,a) \leftarrow N(s,a) + 1$
18 $\quad\quad r \leftarrow r + R[s,a]$
19 **return** *child* (s,a) *of root with max.* $N(s,a)$

candidate a, attach it to the tree and record its reward (ln. 9-13) in the expansion phase. In case action a decides the initial assignment of any flow, we insert a delimiter node with zero reward beforehand to disambiguate where and what flows arrive (ln. 10-11). Third, we simulate coordinating the forecast's remaining flows with randomly selected (valid) decisions in the rollout phase (ln. 14). Its accumulated rewards then give a MC return estimate for the attached node's action value $Q(s,a)$. Due to the random selection of actions and the stochastic prediction of flows, the estimate's variance increases the farther rollouts simulate into the future. This implies a trade-off between the number of iterations, the stability of results and hence the time span over which future demands may be considered. If the additional planning effort cannot be invested under tight budgets, fewer forecast flows (α_{flows}) may be simulated. For $\alpha_{\text{flows}} = 0$, FutureCoord simplifies along the lines of MaVEn-S [19]. Finally, the return estimate and stored rewards are backpropagated upwards the traversed path (ln. 15-18). In each step, we update action values $Q(s,a)$ by the moving average of earlier estimates and this iteration's results. Ideally, if future flow arrivals Z were known, we could evaluate actions by their long-term performance. Averaging action values over simulations of forecasts, instead, gives an expectation over their distribution of interarrival and duration times as well as QoS requirements. If forecasts indeed follow the ground truth distributions of Z, we estimate action values as $\mathbb{E}_{\mathcal{P}_Z}\big[Q(s,a)\big]$, i.e., under the distribution of future flows rather than by the (unknown) Z itself.

V. Trace-Driven Evaluation

In this section, we evaluate FutureCoord using real-world network topologies, traffic patterns and service configurations.

A. Evaluation Setup

1) Simulation Scenarios: Our evaluation considers real-world topologies from SNDlib [47] of up to 50 nodes, including the Abilene network of 12 nodes scattered across the United States. A node's compute capacities cpu_v is chosen uniformly at random between 0.5 and 1.0 units and memory mem_v uniformly from $\{256, 384, 512, 640\}$ units. Link capacities are set to $\lambda_{\{u,v\}} = 9920$ units; link delays $\ell_{\{u,v\}}$ are the propagation delay for nodes' distances. All six offered components $c \in \mathcal{C}$ (e.g. web proxies or web servers) are configured based on real-world performance profiling data from SNDZoo [9]. In detail, we derive fourth-order polynomials to define what compute capacity instances consume. Starting up new instances requires one-time allocations of memory between 64 and 256 units. Flows specify one out of four services $o \in \mathcal{O}$ that differ in length (2-4 components), by their components and by what kind of flows they are typically requested. We consider heterogeneous service configurations and flow demands. First, delay-sensitive flows that must be routed along the shortest route from their ingress to egress, but have low data rates and request few components (service $o^{(1)}$). Second, flows that request services of components that are compute intensive (service $o^{(2)}$) or require large amounts of memory upon instantiation (service $o^{(3)}$). On the other hand, they constrain the max. delay bound of flows less tightly and need not be routed along shortest paths. Finally, we consider high data rate flows of moderate max. delay bounds. They request service $o^{(4)}$ of components with modest resource demands. For each kind of flow, we parameterize normal distributions of flow data rates based on the max. throughput measured during benchmarking for their service [9]. We clip values to be below services' max. throughput and above $1/10$-th of it. To ensure feasible QoS requirements, we consider normally distributed max. delay bounds with mean values proportional to shortest path delays among flows' ingress and egress nodes. Their mean values range from $1\times$ to $4\times$ the flows' shortest path delays. Max. delay bounds are set at least to the shortest path delay. Flows enter the network at times t_f that follow service specific inhomogeneous Poisson processes. The arrival rates of these processes are obtained from averaging real-world traffic traces [47] over sliding windows lengths of 5 minutes. Episode simulate 42 discrete changes of arrival rates with each time period lasting 5 minutes. Thus, we set the time horizon of our evaluation to $T = 43 \cdot 5\text{min} = 12900\text{s}$. Durations δ_f are exponentially distributed with service-specific means. We scale arrival rates and mean durations so that, for example, flows with service $o^{(1)}$ (cmp. mice flows) are more frequent but also depart much sooner than flows of service $o^{(3)}$ (cmp. elephant flows). Where flows enter the network (ingresses) and their traffic exits it (egresses) is also stochastic, service-specific and changes every 5 minutes based on real-world dynamic traffic matrices [47] for the Abilene network.

2) Compared Algorithms & Hyperparameters: We compare our algorithm against one approach from each of the categories heuristics, model-free DRL and planning methods:

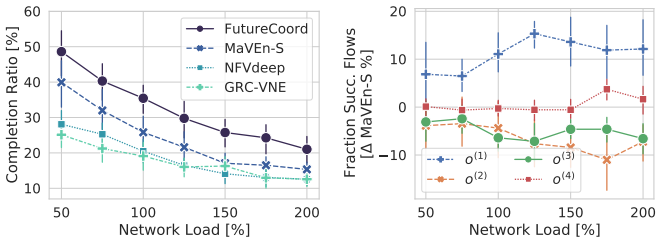
- **GRC-VNE [26]**: a heuristic that ranks nodes by what residual resources they and nearby nodes provide.
- **NFVdeep [35]**: a model-free DRL approach that learns service coordination end-to-end from scratch using DRL.
- **MaVEn-S [19]**: a model-based method that coordinates services using MCTS (cmp. Sec. II).

We set the training of NFVdeep to $2,000,000$ updates per repetition. Both FutureCoord and MaVEn-S perform 500 iterations per decision, although rollouts usually take longer for our work. We truncate rollout simulations to $\alpha_{\text{flows}} = 30$ forecast flows. In our unoptimized, single-core Python simulation, FutureCoord coordinates flows in on the order of tens of seconds (measured on Abilene). Significant runtime improvements could be achieved in future work by *a*) code optimizations *b*) parallel execution *c*) an integration with DRL à la AlphaZero to replace rollout simulations (costlies piece of our algorithm) by learned state value estimations.

3) Execution: Experiments are repeated with 10 different random seeds. Figures show the mean and 95% confidence intervals. For readability, we report flow completion ratios rather than completion rates.

B. Varying Network Resources & Flow Properties

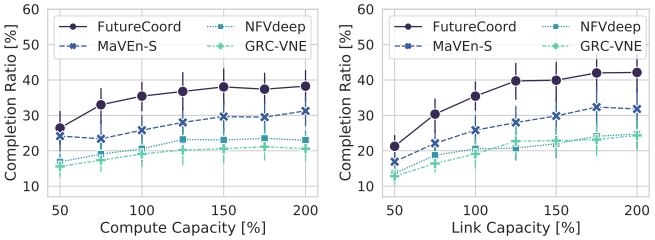
We study the effectiveness of FutureCoord on the Abilene topology. First, we vary the network load, i.e., scale the rate at which each kind of flow arrives proportionally. All other parameters remain unchanged. At higher load, capacities become more and more contested. Effective coordination then involves prioritizing flows that require less capacity. Fig. 3a shows that our work consistently outperforms competing algorithms, in particular at higher loads where an efficient capacity management is essential. Then, it completes up to 51% more flows than its most competitive approach, MaVEn-S. What kind of flows FutureCoord completes is shown Fig. 3b as the fraction of completed services relative to those of MaVEn-S. Indeed, FutureCoord prioritizes flows of modest demands (service $o^{(1)}$) once capacity is scarce. MaVEn-S instead favors resource intensive flows (services $o^{(2)}$ & $o^{(3)}$).



(a) Varying Network Load (b) Flow Priorities

Fig. 3: FutureCoord favors delay sensitive flows at high loads.

Second, we vary provisioned compute and link capacities in Fig. 4a and 4b. Flows enter the network at default rates. In both cases, FutureCoord makes better use of further capacity than competing schemes. Its performance improves considerably for larger link capacities which suggests that load balancing routes is key to complete many flows.



(a) Varying Compute Capacity (b) Varying Link Capacity

Fig. 4: FutureCoord adjusts to varying resource capacities.

Third, we study how performance varies for different (mean) flow data rates. In case flows request lower data rates, capacity is less contested and max. delay bounds become the driving constraint. Fig. 5 shows that our work adjust its coordination policy from preserving capacities along shortest paths to effective resource management under high contention.

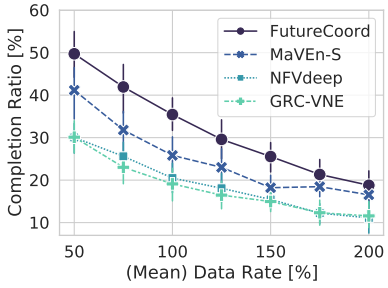
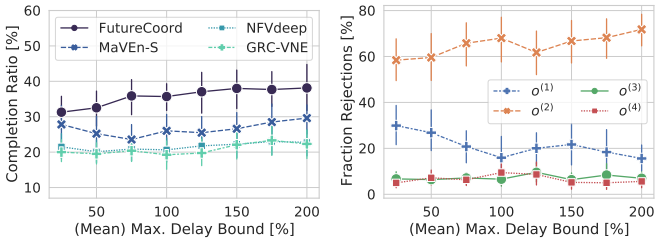


Fig. 5: FutureCoord adjust to varying flow properties.

Finally, we vary the (mean) max. delay bound proportionally for all kinds of flows. Scarce capacities along shortest paths become ever more contested once delay bounds tighten; their efficient usage is critical to complete flows. Vice versa, coordinators may exploit their newly gained flexibility to prioritize flows of otherwise tight max. delay bounds (service $o^{(1)}$) under relaxed conditions. Indeed, FutureCoord adjusts its admission control policy in favor of service $o^{(1)}$ and often dismisses other flows by choice (Fig. 6b) which benefits completion ratios considerably (Fig. 6a).

C. Robustness to Perturbations

In practice, flow property distributions are typically unknown and can only be approximately learned. They then deviate from the ground truth which renders planning with forecasts less precise. To study the extent to which estimation errors are tolerable, we systematically introduce perturbations to the traffic model. First, we perturb the data rate of forecast



(a) Varying Max. Delay Bounds (b) Admission Control Policy

Fig. 6: FutureCoord exploits relaxed max. delay bounds.

flows and vary their expected values in between 25% and 200% of what data rate is expected for future flows (100%). What completion ratios FutureCoord obtains when forecasts under- or overestimate expected flow data rates (Data Rate) or if forecast flows also enter by another episode's pattern (Pattern) is shown in Fig 7. The performance is compared to knowing the ground truth distributions' parameters when planning, as assumed before. Surprisingly, results may even improve slightly ($> 0\%$) in case of prediction errors, but diminish ($< 0\%$) once forecasts overestimate flow data rates. For reference, we also plot competing algorithms unaffected by the perturbations in comparison to FutureCoord's performance given an accurate traffic model (0 %).

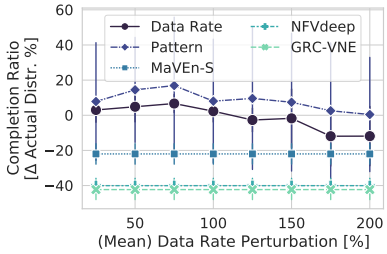
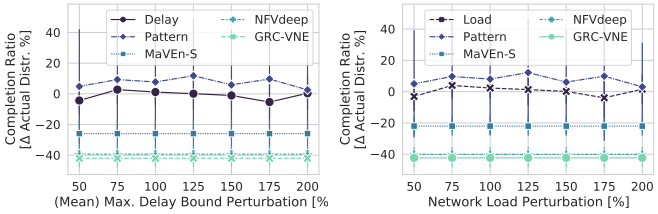


Fig. 7: Flow Data Rate Perturbation

Likewise, we vary the expected max. delay bound and arrival rates of forecast flows in Fig. 8a and Fig. 8b, respectively. Again, FutureCoord consistently outperforms MaVEn-S even in case of severely flawed traffic models and is mostly unaffected by perturbations of its forecasts. This suggests that some general properties may be inferred from forecasts irrespective of possible errors: *a*) what components are offered *b*) what resources they consume *c*) what services are requested and *d*) what components they comprise. Also, it is reasonable to assume that typical tendencies of some flows, for example, specifying tighter max. delay bounds (service $o^{(1)}$) than others (service $o^{(3)}$) can be predicted and are useful when planning service coordination.

D. Scalability

Finally, we evaluate whether FutureCoord performs well on larger real-world topologies from SNDlib [47]. Flows arrive at the same rates as before. We select 12 nodes randomly to serve as possible ingress and egress nodes. Where flows enter



(a) Delay Bound Perturbation (b) Network Load Perturbation

Fig. 8: Performance for estimation errors of forecasts.

and depart is again chosen based on dynamic traffic traces for the Abilene network (12 nodes). Fig. 9 shows FutureCoord's margin of improvement over our most competitive baseline, MaVEn-S, for a varying number of search iterations. On smaller topologies (up to 22 nodes), FutureCoord quickly breaks even with MaVEn-S. To match its performance on larger networks, more search iterations are required. This is likely due to the considered sparse binary feedback and the variance we introduce by sampling forecasts. Estimates become stable when averaged over many iterations (cmp. Sec. IV-B3), but are visited less frequently if more candidates exist. Pruning forecasts to fewer flows α_{flows} helped on larger topologies in our experiments. Still, FutureCoord improves upon our most competitive baseline by a margin of at least 10% for three real-world topologies using few iterations.

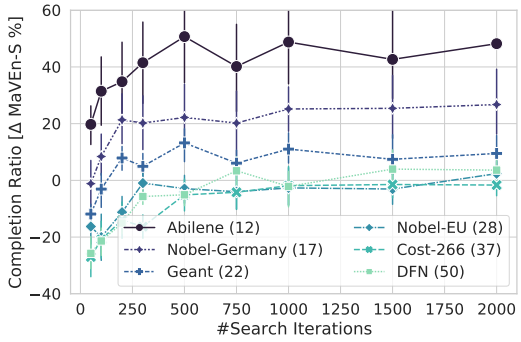


Fig. 9: Margin of improvement over MaVEn-S on various network topologies (#nodes in brackets).

VI. Conclusion

Our proposed model-based AI approach, FutureCoord, uses forecast to maximize long-term flow completion rates. Taking future demands into account, e.g., avoids admitting flows that will occupy resource capacity better used for other flows. Ultimately, our work significantly outperforms existing model-free and model-based approaches in a trace-driven evaluation on real-world network topologies. We believe that FutureCoord is an important step towards using the full potential of model abstractions in AI-based service coordination. In future work, FutureCoord could be integrated with DRL à la AlphaZero to get the best of learning and planning at the same time.

Appendix
Monte Carlo Tree Search

This section briefly outlines the selection, expansion, rollout and backup phases of the MCTS algorithm that we use to coordinate services. Broadly speaking, it explores different candidate coordination decisions using the network model. In doing so, it grows decision trees towards regions of the search space that obtained high accumulated rewards (returns) before or decision areas left unexplored. Each tree node defines a network model state $s \in \mathcal{S}$ and the coordination decision $a \in \mathcal{A}$ that led to it. Search iterations begin at the root tree node (network model state when beginning to plan) and traverse the tree (selection phase). The selection among succeeding tree nodes favors candidates of high returns according to the Upper Confidence Bounds for Trees [48] formula:

$$\underset{(s',a')}{\arg\max}\ Q(s',a') + 2C\sqrt{\frac{2\ln N(s,a)}{N(s',a')}} \tag{2}$$

where (s,a) and (s',a') denote the model states and actions of the parent and successor tree node. Successor tree nodes (s',a') are considered only if decision a' meets all formulated constraints (e.g. max. end-to-end delay bound, cmp. Sec. III). Action values $Q(s,a) = \mathbb{E}\big[r + \gamma V(s') \mid s,a\big]$ formalize what return is expected when action a is taken in state s. This includes the immediate reward r for action a and undiscounted ($\gamma = 1$) state value $V(s')$ of successor model state s'. The number of visits to the parent and child node are stored as $N(s,a)$ and $N(s',a')$. Hyperparameter C quantifies whether the selection among child tree nodes either leans towards exploration or exploitation and is set to 1.41. When C is set to zero, the selection follows branches of highest action values. The traversal terminates once it reaches a tree node with, so far, unvisited children that define valid actions (leaf node). Then, we select among the unvisited (and valid) actions uniformly at random, attach a new child node (s,a) to its predecessor and initialize visit counter $N(s,a)$ to zero (expansion phase). To obtain an initial estimate of the newly added tree node's action value $Q(s,a)$, the model simulation is executed until termination with randomly selected actions (rollout phase). Finally, the simulated return estimate is backpropagated bottom-up along the traversed search path until the root node. Each tree node then increments its visit count $N(s,a)$ and updates $Q(s,a)$ according to the moving average of its former estimate and the upwards propagated simulation return (backup phase). Once the computational budged dedicated towards planning is exhausted, we select the root node's child and its action that was visited most often.

Acknowledgments

We thank Nils Rudminat for his valuable contributions to our first prototype. This work has received funding from the German Research Foundation (DFG) within the Collaborative Research Center "On-The-Fly Computing" (SFB 901).

References

[1] J. Halpern and C. Pignataro. Service function chaining (sfc) architecture. Accessed on 05.09.2021. [Online]. Available: https://www.rfc-editor.org/info/rfc7665

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

[3] ITU. Architectural framework for machine learning in future networks including imt-2020 (y.3172). Accessed on 14.09.2021. [Online]. Available: https://www.itu.int/rec/T-REC-Y.3172/en

[4] C. Berner, G. Brockman, B. Chan, V. Cheung, P. Dębiak, C. Dennison, D. Farhi, Q. Fischer, S. Hashme, C. Hesse *et al.*, "Dota 2 with large scale deep reinforcement learning," *arXiv:1912.06680*, 2019.

[5] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[6] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing Atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[7] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger, "Deep reinforcement learning that matters," in *AAAI Conference on Artificial Intelligence*, vol. 32, no. 1, 2018.

[8] S. Van Rossem, W. Tavernier, D. Colle, M. Pickavet, and P. Demeester, "Profile-based resource allocation for virtualized network functions," *IEEE Transactions on Network and Service Management*, vol. 16, no. 4, pp. 1374–1388, 2019.

[9] M. Peuster, S. Schneider, and H. Karl, "The softwarised network data zoo," in *2019 15th International Conference on Network and Service Management (CNSM)*. IEEE, 2019, pp. 1–5.

[10] A. Bayati, V. Asghari, K. Nguyen, and M. Cheriet, "Gaussian process regression based traffic modeling and prediction in high-speed networks," in *2016 IEEE Global Communications Conference (GLOBECOM)*. IEEE, 2016, pp. 1–7.

[11] T. Subramanya and R. Riggio, "Machine learning-driven scaling and placement of virtual network functions at the network edges," in *2019 IEEE Conference on Network Softwarization (NetSoft)*. IEEE, 2019, pp. 414–422.

[12] C. Hardegen, B. Pfülb, S. Rieger, A. Gepperth, and S. Reißmann, "Flow-based throughput prediction using deep learning and real-world network traffic," in *2019 15th International Conference on Network and Service Management (CNSM)*. IEEE, 2019, pp. 1–9.

[13] K. Qu, W. Zhuang, X. Shen, X. Li, and J. Rao, "Dynamic resource scaling for vnf over nonstationary traffic: A learning approach," *IEEE Transactions on Cognitive Communications and Networking*, vol. 7, no. 2, pp. 648–662, 2020.

[14] J. Kim and G. Hwang, "Adaptive bandwidth allocation based on sample path prediction with gaussian process regression," *IEEE Transactions on Wireless Communications*, vol. 18, no. 10, pp. 4983–4996, 2019.

[15] G. Dulac-Arnold, N. Levine, D. J. Mankowitz, J. Li, C. Paduraru, S. Gowal, and T. Hester, "Challenges of real-world reinforcement learning: definitions, benchmarks and analysis," *Machine Learning*, pp. 1–50, 2021.

[16] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel *et al.*, "Mastering chess and shogi by self-play with a general reinforcement learning algorithm," *arXiv preprint arXiv:1712.01815*, 2017.

[17] G. Chaslot, S. Bakkes, I. Szita, and P. Spronck, "Monte-Carlo Tree Search: A new framework for game ai." *AIIDE*, vol. 8, pp. 216–217, 2008.

[18] O. Soualah, M. Mechtri, C. Ghribi, and D. Zeghlache, "An efficient algorithm for virtual network function placement and chaining," in *2017 14th IEEE Annual Consumer Communications & Networking Conference (CCNC)*. IEEE, 2017, pp. 647–652.

[19] S. Haeri and L. Trajković, "Virtual network embedding via monte carlo tree search," *IEEE Trans. on Cybernetics*, vol. 48, no. 2, pp. 510–521, 2017.

[20] G. Zheng, C. Wang, W. Shao, Y. Yuan, Z. Tian, S. Peng, A. K. Bashir, and S. Mumtaz, "A single-player monte carlo tree search method combined with node importance for virtual network embedding," *Annals of Telecommunications*, pp. 1–16, 2020.

[21] S. Werner, "Futurecoord GitHub repository," https://github.com/CN-UPB/FutureCoord (August 22, 2021), 2021.

[22] A. Laghrissi and T. Taleb, "A survey on the placement of virtual resources and virtual network functions," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 2, pp. 1409–1434, 2018.

[23] R. Bruschi, A. Carrega, and F. Davoli, "A game for energy-aware allocation of virtualized network functions," *Journal of Electrical and Computer Engineering*, vol. 2016, 2016.

[24] R. Cohen, L. Lewin-Eytan, J. S. Naor, and D. Raz, "Near optimal placement of virtual network functions," in *2015 IEEE Conference on Computer Communications (INFOCOM)*. IEEE, 2015, pp. 1346–1354.

[25] M. Yu, Y. Yi, J. Rexford, and M. Chiang, "Rethinking virtual network embedding: Substrate support for path splitting and migration," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 17–29, 2008.

[26] L. Gong, Y. Wen, Z. Zhu, and T. Lee, "Toward profit-seeking virtual network embedding algorithm via global resource capacity," in *IEEE INFOCOM 2014-IEEE Conference on Computer Communications*. IEEE, 2014, pp. 1–9.

[27] Q. Sun, P. Lu, W. Lu, and Z. Zhu, "Forecast-assisted nfv service chain deployment based on affiliation-aware vnf placement," in *2016 IEEE Global Communications Conf. (GLOBECOM)*. IEEE, 2016, pp. 1–6.

[28] X. Fei, F. Liu, H. Xu, and H. Jin, "Adaptive vnf scaling and flow routing with proactive demand prediction," in *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*. IEEE, 2018, pp. 486–494.

[29] X. Zhang, C. Wu, Z. Li, and F. C. Lau, "Proactive vnf provisioning with multi-timescale cloud resources: Fusing online learning and online optimization," in *IEEE INFOCOM 2017-IEEE Conference on Computer Communications*. IEEE, 2017, pp. 1–9.

[30] H. Tang, D. Zhou, and D. Chen, "Dynamic network function instance scaling based on traffic forecasting and vnf placement in operator data centers," *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 3, pp. 530–543, 2018.

[31] N. C. Luong, D. T. Hoang, S. Gong, D. Niyato, P. Wang, Y.-C. Liang, and D. I. Kim, "Applications of deep reinforcement learning in communications and networking: A survey," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 4, pp. 3133–3174, 2019.

[32] Z. Yan, J. Ge, Y. Wu, L. Li, and T. Li, "Automatic virtual network embedding: A deep reinforcement learning approach with graph convolutional networks," *IEEE Journal on Selected Areas in Communications*, vol. 38, no. 6, pp. 1040–1057, 2020.

[33] W. Mao, L. Wang, J. Zhao, and Y. Xu, "Online fault-tolerant vnf chain placement: A deep reinforcement learning approach," in *2020 IFIP Networking Conference (Networking)*. IEEE, 2020, pp. 163–171.

[34] S. Schneider, A. Manzoor, H. Qarawlus, R. Schellenberg, H. Karl, R. Khalili, and A. Hecker, "Self-driving network and service coordination using deep reinforcement learning," in *2020 16th International Conference on Network and Service Management (CNSM)*. IEEE, 2020, pp. 1–9.

[35] Y. Xiao, Q. Zhang, F. Liu, J. Wang, M. Zhao, Z. Zhang, and J. Zhang, "Nfvdeep: Adaptive online service function chain deployment with deep reinforcement learning," in *Proceedings of the International Symposium on Quality of Service*, 2019, pp. 1–10.

[36] J. Pei, P. Hong, M. Pan, J. Liu, and J. Zhou, "Optimal vnf placement via deep reinforcement learning in sdn/nfv-enabled networks," *IEEE Journal on Selected Areas in Communications*, vol. 38, no. 2, pp. 263–278, 2019.

[37] R. Solozabal, J. Ceberio, A. Sanchoyerto, L. Zabala, B. Blanco, and F. Liberal, "Virtual network function placement optimization with deep reinforcement learning," *IEEE Journal on Selected Areas in Communications*, vol. 38, no. 2, pp. 292–303, 2019.

[38] P. T. A. Quang, Y. Hadjadj-Aoul, and A. Outtagarts, "A deep reinforcement learning approach for vnf forwarding graph embedding," *IEEE Transactions on Network and Service Management*, vol. 16, no. 4, pp. 1318–1331, 2019.

[39] Z. Xu, J. Tang, J. Meng, W. Zhang, Y. Wang, C. H. Liu, and D. Yang, "Experience-driven networking: A deep reinforcement learning based approach," in *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*. IEEE, 2018, pp. 1871–1879.

[40] L. Gu, D. Zeng, W. Li, S. Guo, A. Y. Zomaya, and H. Jin, "Intelligent vnf orchestration and flow scheduling via model-assisted deep reinforcement learning," *IEEE Journal on Selected Areas in Communications*, vol. 38, no. 2, pp. 279–291, 2019.

[41] O. Soualah, I. Fajjari, N. Aitsaadi, and A. Mellouk, "A batch approach for a survivable virtual network embedding based on monte-carlo tree

search," in *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*. IEEE, 2015, pp. 36–43.

[42] S. Dräxler, M. Peuster, M. Illian, and H. Karl, "Generating resource and performance models for service function chains: the video streaming case," in *2018 4th IEEE Conference on Network Softwarization and Workshops (NetSoft)*. IEEE, 2018, pp. 318–322.

[43] C. N. C. Foundation. Production-grade container orchestration. Accessed on 05.09.2021. [Online]. Available: https://kubernetes.io/

[44] S. Schneider, N. P. Satheeschandran, M. Peuster, and H. Karl, "Machine learning for dynamic resource allocation in network function virtualization," in *2020 6th IEEE Conference on Network Softwarization (NetSoft)*. IEEE, 2020, pp. 122–130.

[45] H. Mao, S. B. Venkatakrishnan, M. Schwarzkopf, and M. Alizadeh, "Variance reduction for reinforcement learning in input-driven environments," *arXiv preprint arXiv:1807.02264*, 2018.

[46] J. Son, A. V. Dastjerdi, R. N. Calheiros, X. Ji, Y. Yoon, and R. Buyya, "Cloudsimsdn: Modeling and simulation of software-defined cloud data centers," in *2015 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*. IEEE, 2015, pp. 475–484.

[47] S. Orlowski, M. Pióro, A. Tomaszewski, and R. Wessäly, "SNDlib 1.0–Survivable Network Design Library," in *Proceedings of the 3rd International Network Optimization Conference (INOC 2007), Spa, Belgium*, April 2007, http://sndlib.zib.de, extended version accepted in Networks, 2009. [Online]. Available: http://www.zib.de/orlowski/Paper/OrlowskiPioroTomaszewskiWessaely2007-SNDlib-INOC.pdf.gz

[48] L. Kocsis and C. Szepesvári, "Bandit based monte-carlo planning," in *European Conf. on machine learning*. Springer, 2006, pp. 282–293.