Deep Reinforcement Learning for Adaptive Traffic Engineering in Satellite Constellation Networks

1st Manuel M. H. Roth
Inst. of Communications & Navigation
German Aerospace Center (DLR)
Oberpfaffenhofen, Germany
manuel.roth@dlr.de

2nd Thomas Jerkovits
Inst. of Communications & Navigation
German Aerospace Center (DLR)
Oberpfaffenhofen, Germany
thomas.jerkovits@dlr.de

3rd Anupama Hegde
Inst. of Communications & Navigation
German Aerospace Center (DLR)
Oberpfaffenhofen, Germany
anupama.hegde@dlr.de

4th Thomas Delamotte
Chair of Signal Processing
University of the Bundeswehr Munich
Neubiberg, Germany
thomas.delamotte@unibw.de

5th Andreas Knopp
Chair of Signal Processing
University of the Bundeswehr Munich
Neubiberg, Germany
andreas.knopp@unibw.de

***Abstract*—With increasing demand for broadband services provided by satellite constellation networks, routing and traffic management have become increasingly relevant topics. To enable global coverage and end-to-end connectivity, these systems rely on inter-satellite links to span space-borne networks. To fulfill the quality of service requirements of significant network loads, the traffic load needs to be balanced optimally. As rule-based techniques to solve the underlying multi-commodity flow problem are too complex to comply with on-board processing power limitations, specifically tailored light-weight solutions are required. To this end, we propose an adaptive traffic engineering approach based on deep reinforcement learning. We explore a policy-based approach for a flexible flow allocation between candidate paths. We compare the schemes with state-of-the-art benchmarks based on heuristics, and an optimal benchmark using linear programming. The results highlight that the proposed scheme is able to approximate optimal solutions to the multi-commodity flow problem and can learn suitable policies for diverse sets of paths. Moreover, after training, the approach exhibits low complexity in inference, and is thus well-suited to be included in the controller logic of in-space distributed software defined networks.**

***Index Terms*—routing, traffic engineering, deep reinforcement learning, satellite networks**

I. Introduction

To support increasing bandwidth requirements, Low Earth Orbit (LEO) Satellite Constellation Networks (SCNs) require specifically tailored routing and network management protocols [1]. Different architectures have been proposed to enable effective load-balancing approaches. For instance, space-borne distributed Software Defined Networking (SDN) has been considered [2]. To maximize the throughput within such networks, controllers have to make informed decisions which use the network resources efficiently. This optimization builds upon the underlying Multi-Commodity Flow Problem (MCFP). For a given set of commodities with different source-destination pairs and an associated demand, an optimal routing configuration must be determined. Computing this optimal routing configuration is computationally complex, the integer MCFP is NP-hard in general [3]. While Linear Programming (LP) and Dynamic Programming (DP) approaches can be used as solvers [4], they often require significant computational resources [5]. Various schemes have been proposed using viable heuristics to reduce these processing demands [4]. In space networks, the limited on-board processing power of satellites typically makes running exhaustive algorithms infeasible. Instead, heuristics for LP have been proposed [6].

Supported by the European Space Agency (ESA), activity code 5A.079

In real-world systems however, flows typically arrive concurrently and subsequently. Making choices iteratively, or stepwise, is challenging. Greedy approaches, such as choosing the least-congested path for each incoming flow, rarely result in optimal solutions as will be shown in our evaluation. Instead, effective schemes have to anticipate future flow patterns and make choices accordingly to approximate MCFP solutions.

Deep Reinforcement Learning (DRL) is an effective approach to handle overlapping sub-problems through the maximization of expected future rewards. The technique relies on an agent interacting with its environment, enabling it to develop policies that balance immediate performance and expected long-term impact. Unlike heuristics, DRL can learn from experience and adapt to changing network conditions. As it deduces policies directly from data, it has the capacity to discover non-obvious traffic patterns and adjust accordingly. An autonomous adaptation to LEO constellation dynamics through real-time policy learning and optimization is uniquely enabled. Once trained, inference is typically of low complexity. DRL-based approaches to approximate the MCFP have also been proposed for SCNs [7]. Further investigations into DRL-based routing for SCNs have also been made [8]. In contrast to these works, which rely on hop-based action spaces, we propose a path-based approach. The agent is distributes flows on candidate paths while trying to approximate solutions to the MCFP.

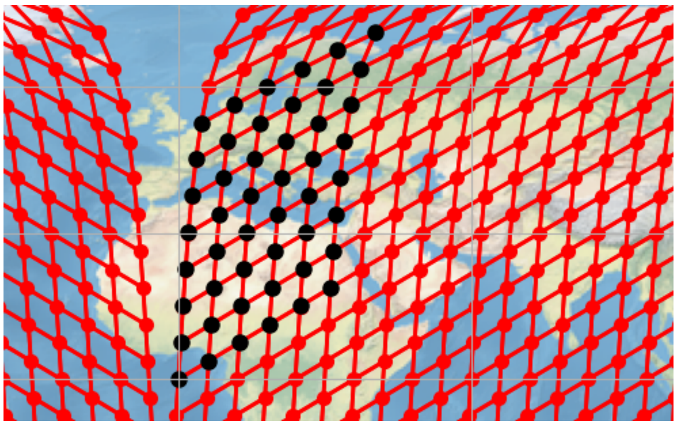

Fig. 1. Grid of polar satellite constellation, inter-connected via ISLs. Nodes highlighted in black form rectangular sub-network, e.g. as in the context of space-borne distributed SDN.

The contributions of this work include:

- A novel path-based DRL approach which distributes flows on corresponding candidate paths. The approach leverages Soft Actor Critic (SAC) (as in [9], [10]).
- An evaluation of the proposed approach in two scenarios: with a static flow set, and with random flow sets.
- A comparison of the approach with various state-of-the-art schemes and a benchmark solution.

II. Reference System

A. Network & Connectivity

We consider an exemplary grid network based on the structure of large LEO SCNs. In these systems, the satellites are assumed to be inter-connected via Intersatellite Links (ISLs). We focus in our investigation on sub-networks of the resulting space-borne network grid structure. Such a sub-division of the network into smaller units can be achieved with distributed SDN [2]. An example of such a network is shown in Figure 1. It is important to note that there is an inherent path diversity to such SCNs: typically there are multiple viable paths of similar end-to-end characteristics. So, traffic engineering protocols can significantly impact the amount of supported network load by distributing traffic efficiently.

B. Multi-Commodity Flow Problem in Static Networks

The overall objective is to support as many incoming flows as possible without saturating links. To this end, we aim to approximate solutions to the MCFP. The MCFP provides a framework for modeling and solving the network flow distribution when multiple traffic demands (commodities) must be routed through a network with limited capacity.

Let $G = (V, E)$ be a directed graph representing the network, where V is the set of nodes and E is the set of edges. Each edge $(i, j) \in E$ has a capacity $c(i, j) \in \mathbb{R}_{>0}$.

Let $T = \{1, 2, \ldots, N\}$ be the index set of commodities, where N is the total number of commodities. Each commodity is identified by its integer index $t \in T$. For each commodity $t \in T$ we have

- $x_t \in V$ is the source node,
- $y_t \in V$ is the target (or destination) node,
- $d_t \in \mathbb{R}_{>0}$ is the demand to be routed from x_t to y_t.

Let $f_t(i, j) \in \mathbb{R}_{\geq 0}$ be the flow of commodity t on edge (i, j). The total flow on edge (i, j) is

$$f(i, j) = \sum_{t \in T} f_t(i, j).$$

Our objective is to minimize the maximum utilization among all edges in the network. The *utilization* $u(i, j)$ of edge (i, j) is defined as

$$u(i, j) = \frac{f(i, j)}{c(i, j)}.$$

Thus, the objective function is

$$\min \max_{(i,j) \in E} u(i, j).$$

The MCFP can be formulated as a linear program and solved efficiently using LP techniques. The problem is subject to the following constraints.

Flow conservation constraints: For all $t \in T$ and $i \in V$

$$\sum_{(i,j) \in E} f_t(i, j) - \sum_{(j,i) \in E} f_t(j, i) = \begin{cases} d_t, & \text{if } i = x_t, \\ -d_t, & \text{if } i = y_t, \\ 0, & \text{otherwise.} \end{cases}$$

Capacity constraints: For all edges $(i, j) \in E$,

$$f(i, j) \leq c(i, j).$$

Non-negativity constraints:

$$f_t(i, j) \geq 0, \quad \forall t \in T, \ \forall (i, j) \in E.$$

In the integer version of the MCFP, flows are restricted to integer values, i.e.

$$f_t(i, j) \in \mathbb{Z}_{\geq 0}, \quad \forall t \in T, \ \forall (i, j) \in E.$$

The integer MCFP is more representative of real-world network flows, such as those enabled by SDN, where packets are the minimum splitting unit. This problem is NP-hard in general [3].

We use the solution to the MCFP as an optimal benchmark in deterministic scenarios where all future commodities are assumed to be known beforehand. However, in realistic routing scenarios, the exact set of flows is typically unpredictable. Although characteristic traffic patterns may emerge, the combinatorial space of possible flow configurations grows exponentially with network size, making exhaustive MCFP computations intractable. Hence, adaptive strategies rather than static, pre-computed solutions are required. A dynamic formulation of the MCFP exists where elements such as edges, capacities, flows, and demands can vary over time steps, making it particularly suitable for time-dependent flow scenarios. In this investigation, however, we focus on the static case, which provides a reasonable approximation for scenarios with rapidly incoming flow requests.

C. Approximating Least Congested Path Computation

To approximate the least congested path, we reformulate the min-max problem as a min-sum problem. For each edge $(i,j) \in E$ with utilization $u(i,j)$, we define an auxiliary weight $w(i,j) = \exp(u(i,j))$. This transformation is inspired by the Jacobian logarithm approximation, where

$$\log(\exp(a) + \exp(b)) = \max(a,b) + \log(1 + \exp(-|a-b|))$$

$$\approx \max(a,b).$$

To find the path $p \subset E$ that minimizes the sum of these auxiliary weights

$$\sum_{(i,j)\in p} w(i,j) = \sum_{(i,j)\in p} \exp(u(i,j)),$$

we can run Dijkstra's algorithm. The logarithm of this sum

$$\log(\sum_{(i,j)\in p} \exp(u(i,j))) \approx \max_{(i,j)\in p} u(i,j),$$

minimizes the sum of exponentials. Thus, it provides an effective approximation for minimizing the maximum edge utilization along the path. Therefore, we use this approximation as a link weight for a baseline method called "least-congested-first". It is also used in the computation of candidate paths for the proposed DRL approach.

III. Deep Reinforcement Learning Approach

DRL techniques have been recognized as effective alternatives for problems where heuristics may perform poorly and DP becomes computationally expensive [5]. In the context of traffic engineering for SCNs, DRL is of particular interest as low-complexity decision-making based on dynamically evolving states is required. As mentioned, several schemes have been proposed for RL-based routing, many of them hop-based. A crucial drawback of this method is that the path finding problem has to be learned as well. This adds complexity to the original problem we want the agent to focus on: iteratively approximating the MCFP. Instead of learning path finding, we propose to leverage existing path selection algorithms to generate sets of candidate paths. The core assumption to this approach is that an optimal solution can be approximated by splitting incoming flows on these candidate paths. In principle, the idea is to learn underlying traffic characteristics which enables the anticipation of bottlenecks. Consequently, higher network loads can be supported thanks to informed decision-making.

A. Architecture

As the agent is supposed to learn suitable policies, a policy-based architecture has to be considered. To avoid variable action space sizes, we enforce a limit to the candidate path set. For the sake of simplicity, we assume any flow fraction possible. The resulting action space is thus continuous representing flow fractions among these paths. It is conceptually similar to a probability simplex.

To handle the action space and the stochastic policies, we propose a SAC [9], [10] architecture. SAC is an off-policy actor-critic algorithm which maximizes the expected return while also maximizing entropy to encourage exploration. A stochastic policy is learned, as well as value functions, and a temperature parameter to balance exploration and exploitation. It consists of a policy network (π_ϕ), and two Q-value networks ($Q_{\theta_1}, Q_{\theta_2}$). The temperature parameter α adjusts the weight of the entropy term. The objective function $J_Q(\theta_i)$ for updating the soft Q-function parameters θ_1, θ_2 for state s_t, action a_t, at time step t is the minimization of the soft Bellman residual.

$$J_Q(\theta_i) = \mathbb{E}_{(s_t,a_t)\sim\mathcal{D}} \left[\frac{1}{2} \left(Q_{\theta_i}(s_t,a_t) - \hat{Q}_{\theta_i}(s_t,a_t) \right)^2 \right]$$

$\hat{Q}_{\theta_i}(s_t,a_t)$ represents the estimated target. $\mathcal{D}$ denotes the replay buffer or a distribution of previously sampled states and actions. The policy parameters ϕ are updated by minimizing the divergence between the policy and the exponential of the soft Q-funtion using the entropy temperature parameter α,

$$J_\pi(\phi) = \mathbb{E}_{s_t\sim\mathcal{D}} \left[\mathbb{E}_{a_t\sim\pi_\phi} \left[\alpha \log \pi_\phi(a_t \mid s_t) - Q_\theta(s_t,a_t) \right] \right].$$

This encouragement of exploration aims to avoid premature convergence to sub-optimal policies, which is a concern in the given scenario.

B. Environment

The environment consists of the presented grid network graph G. At each step t, an incoming flow request must be routed through the network. This request is represented by a tuple consisting of a source node, target node, and demand, denoted as (x_t, y_t, d_t), respectively. Using efficient path-finding algorithms, a set of k candidate paths from source node x_t to target node y_t is pre-generated. The source and target node of flows are sampled from a fixed heat-map. The demand of a flow is sampled from a uniform distribution. Link utilization is updated according to the flow allocations.

C. Action Space

The task of the agent is to split the incoming flow request among the pre-generated candidate paths. Thus, we define the action space as a vector of length k over $\mathbb{R}_{\geq 0}$, where k corresponds to the number of candidate paths pre-generated for the current flow request. To ensure the action represents valid fractions of the flow, the agent's chosen action vector is normalized such that its entries sum to 1. This normalization ensures that the flow is divided proportionally among the candidate paths based on the agent's output.

D. Set of Candidate Paths

In order for the agent to learn effective policies, it is important that the action space and its impact in terms of reward are varied. Since the key assumption of approximating optimal flow distribution relies on this set, its design is critical. Naturally, the set should include less congested paths to offer viable options. However, the paths should only minimally overlap, so that the effects of choices are distinguishable. We

TABLE I
CONSIDERED SETS OF CANDIDATE PATHS. THE OVERLAP INDICATES THE PROPORTION OF NODES A PATH CAN SHARE WITH OTHER PATHS.

Name	Set (a)	Set (b)	Set (c)
Path 1	shortest path in hops	least-weight*	least-weight*
Path 2	no edge with path 1	50% overlap	70% overlap
Path 3	least-weight*	50% overlap	70% overlap
Path 4	2^{nd} least-weight*	50% overlap	70% overlap
$*$: weight on hop is $w(i,j) = \exp(u(i,j))$			

investigate three sets of candidate paths which are listed in Table I. As indicated, training with different sets resulted in different behaviours.

E. State Space

1) One-Hot Encoding: To enable coherence with the action space, the observation space of the agent has to include the path of the considered candidate routes, which are the actions. Thus, the agent can map topological characteristics to actions. Since the path size is variable with a maximum viable path length, we have to embed the information. A simple approach is to consider edge-based one-hot encoding to represent paths. In addition, we include path characteristics in the observation. Namely, the maximum, average, and variance of the current link utilization on the path. Moreover, we include the demand of the flow as well as the current link utilization matrix. The state space vector consists of real-valued elements, and its length, denoted by S_1, is given by

$$S_1 = k \cdot (|E| + 3) + 1 + |E|, \tag{1}$$

where k is the number of candidate paths, and $|E|$ is the number of edges in the network graph. The 3 comes from the three path characteristics and the 1 is for the current demand.

2) Histogram State Space: An alternative state space represents the system using a histogram of the current edge utilization, k histograms for the utilization of the candidate paths, and the demand. The length of this state representation, denoted by S_2, is given by

$$S_2 = (k + 1) \cdot H + 1, \tag{2}$$

where H is the length of the histogram vector. Each histogram captures the relative appearance of utilization values on the edges, represented as a list of values between 0 and 1, where the entries sum to 1. Importantly, the length of S_2 is independent of the number of edges in the graph, making it scalable to different graph sizes. The value of H depends on the number of bins b used for discretizing utilization values, with larger b providing a finer-grained representation but increasing the size of H and, consequently, S_2.

We intend to compare this *histogram-based state space* with the *one-hot encoding state space* under different scenarios.

F. Reward

We shape the rewards according to the intended characteristics of the MCFP. Correctly allocated flows are rewarded, while the saturation of links is punished. We also include the highest new link utilization introduced by the flow allocation. The reward function at time step $t+1$ is defined as

$$r_{t+1} = -\,\xi \sum_{v=1}^{k} \sum_{(i,j)\in p_t^{(v)}} \big(1 + \frac{u(i,j)}{c(i,j)}\big) \cdot \mathbb{1}[u(i,j) > c(i,j)] \tag{3}$$
$$+\,\nu\big(1 \;-\; \max_{(i,j)\in \cup_{v=1}^{k} p_t^{(v)}} u(i,j)\big) \tag{4}$$
$$+\,\mu \sum_{v=1}^{k} \sum_{(i,j)\in p_t^{(v)}} d_t a_t^{(v)} \cdot \mathbb{1}[u(i,j) \le c(i,j)]\,, \tag{5}$$

where

- $p_t^{(1)}, p_t^{(2)}, \ldots, p_t^{(k)}$ are the k candidate paths at time step t, and each $p_v^{(t)}$ is a set of edges.
- $a_t = (a_t^{(1)}, a_t^{(2)}, \ldots, a_t^{(k)}) \subseteq \mathbb{R}^k_{\geq 0}$ is the agent's action at time step t, with $a_t^{(v)}$ being the fraction of demand d_t allocated to path $p_t^{(v)}$, satisfying $\sum_{v=1}^{k} a_t^{(v)} = 1$.
- $\mathbb{1}[\cdot]$ is the indicator function, which equals 1 if the condition in the argument is true and 0 otherwise.
- μ, ν, and ξ are positive weight factors to balance the components of the reward.

The term in (3) penalizes paths with edges whose utilization exceeds capacity, scaled by ξ. Similarly, the term in (4) incentivizes minimizing the maximum utilization across all edges, weighted by ν. Lastly, the term in (5) rewards successful allocation of demand to paths without exceeding capacity, scaled by μ.

If the set of candidates consists of lesser congested paths, the agent will only encounter punishments at high network loads. Due to the scope of this work, the effects of various reward shapes are not investigated.

G. Complexity

Once trained the complexity of the proposed approach consists of the complexity of computing the candidate paths and the complexity of model inference, i.e.

$$\mathcal{T}_{\text{deploy}} = \mathcal{T}_{\text{paths}} + \mathcal{T}_{\text{inference}}$$

Using a k-shortest paths algorithm [11], a complexity in time of

$$\mathcal{T}_{\text{paths}} = O(|E| + |V| \log |V| + k).$$

can be achieved. As before, $|E|$ denotes the number of edges, and $|V|$ the number of nodes. Since the candidate paths may be computed using a mix of path-finding algorithms, the resulting complexity is slightly higher, but assumed to be in a similar domain. The inference consists of a forward pass through the actor network. So, a matrix multiplication with the layers of the trained neural network. L_{actor} denotes the number of layers, and $n_{actor,l}$ the neurons per layer, i.e.

$$\mathcal{T}_{\text{inference}} = O\big(L_{actor} n^2_{actor,l}\big).$$

Training complexity, on the other hand, is much higher. It is the product of the batch size B, and the time complexity of a

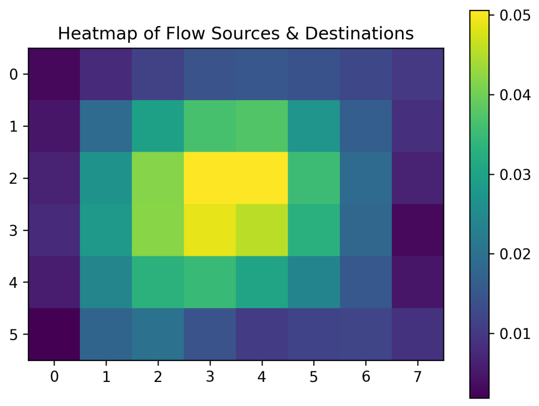

Fig. 2. Considered heat-map of traffic sources and destinations. The probabilities to act as a source or destination of the considered 48 nodes are shown. Node index represented by row and column.

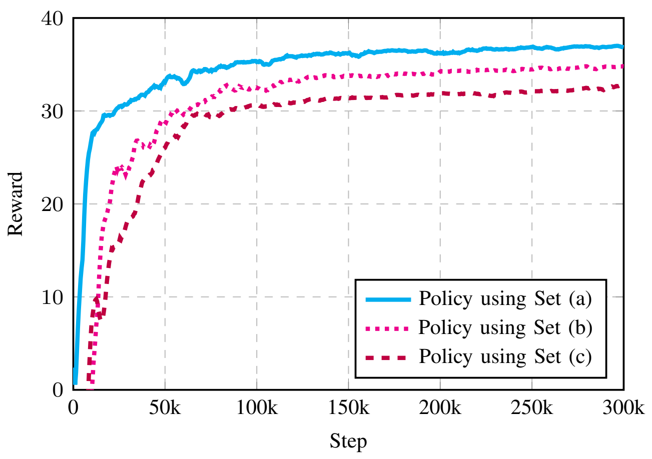

Fig. 3. Comparison of mean episode rewards for candidate path algorithms in scenario using deterministic set of flows.

forward and backward pass through the actor network ($\mathcal{T}_{actor}$) and the two critic networks ($2\mathcal{T}_{critic}$) per sample, hence

$$\mathcal{T}_{\text{training}} = O(B\,(\mathcal{T}_{\text{actor}} + 2\mathcal{T}_{\text{critic}})).$$

IV. Performance Comparison

A. Simulative Environment

The environment is implemented in Python using the `Gymnasium` library [12], with SAC algorithms provided by `Stable-Baselines3` [13]. The underlying graph of the 2D grid network and path-finding routines are implemented using the Python library `NetworkX` [14].

For this investigation, we consider a network of 48 satellite nodes, in a 8×6 arrangement, similar to the cluster shown in Figure 1. Flows from a source to a destination node are sampled randomly according to the heat-map shown in Figure 2. The demand, i.e. the utilization of the flow, is uniformly sampled from the interval $[0.1, 0.5]$. In each step, a new flow is either read from a pre-existing set or generated. Then, according candidate paths are computed and provided as part of the observation along with path characteristics as described in III-B with $k = 4$. We consider sets of $N = 50$ flows, as the resulting load proves challenging for rule-based approaches, but optimal solutions which do not surpass the maximum capacity on any link are still possible in many cases. The used hyperparameters are summarized in Table II.

B. Comparison of Candidate Path Selection

To evaluate the effects of the considered path selection schemes (in Table I), the highest achieved mean reward during training is investigated. As the scenarios are varied, it is difficult to make universal statements about the performance of the path selection sets. In Figure 3, the mean rewards while training on a deterministic set of flows for 300,000 iterations is shown. For these flow requests, similar rewards were achieved by the different methods. Although path set (a) provided slightly higher rewards, its actual performance in inference was worse than the solution using path set (b). In general, the result strongly depends on the considered flow set. For randomly sampled flow sets in particular, candidate path set (b) yields the best performance. Candidate set (c) is outperformed by set (b) in almost all scenarios. Therefore, we assume that the increased path diversity of set (b) is typically preferable, even if there are specific scenarios where set (a) can achieve similar performance.

TABLE II
Hyperparameters of the SAC model for One-Hot Encoding and Histogram Policies using $k = 4$ precomputed paths.

Hyperparameter	**One-Hot Encoding**	**Histogram Policy**
Algorithm	SAC	SAC
Policy	MlpPolicy	MlpPolicy
State Space Size	$S_1 = 833$ (see (1))	$S_2 = 51$ (see (2))
Policy Architecture	$[512, 256, 128]$	$[128, 128]$
Critic Architecture	$[512, 256, 128]$	$[128, 128]$
Action Space Size	$k = 4$	$k = 4$
Activation Function	ReLU	ReLU
Learning Rate	$1 \cdot 10^{-3}, 1 \cdot 10^{-4}$	$1 \cdot 10^{-3}$
Buffer Size	1,000,000	1,000,000
Batch Size	512	512
Target Update Rate	0.01	0.01
Discount Factor	0.999	0.999
Entropy Coefficient	Automatic	Automatic

C. Baseline Methods

The methods, to which the proposed approach is compared, are supposed to serve as lower, upper, and state-of-the-art benchmarks.

1) Shortest-Path First (SPF) Algorithm: We include a delay-based shortest-path-first approach as a lower benchmark. While the approach is link load-agnostic, it introduces the lowest overhead in terms of network load. As it relies on the shortest paths only, it introduces the lowest overall network utilization since no additional hops are used. The flow requests are not split in this approach.

2) Least-Congested-Path First (LCF) Algorithm: Essentially the same approach as the previous method. However, the edge weights represent link loads. Therefore, the lowest-cost path is the path with the lowest aggregated link load. The flow requests are also not split in this approach. This approach is important to show that the agent is able to learn a better policy than just always choosing the currently least-congested path.

3) Step-wise LP: It is possible to find an allocation for each incoming flow using LP. The flow is allocated to different paths based on the current state of the network. However, the approach does not consider how its decisions may impact future network states. Therefore, it can be outperformed by a well-trained approach which is familiar with underlying traffic patterns and able to choose paths accordingly.

4) Equal Multi-Path (EMP) Algorithm: To evaluate the impact of the trained policies, we also include a comparison to an approach which distributes incoming flows uniformly between the candidate paths. It is included in the comparison in Section IV-D2.

5) Upper Benchmark - Ideal Allocation: If we provide an LP solver with the complete set of incoming flows, it can compute the optimal flow allocation accordingly. It thus represents the upper benchmark the RL-based algorithm is trying to approach. Naturally, this approach is purely theoretical as all incoming flows are known and allocated simultaneously. Effectively, it operates with perfect knowledge of all future flow requests. It is included in the comparison in Section IV-D1.

D. Performance Comparison

1) Deterministic Set of Flows: Firstly, we investigate the convergence and performance of the considered approach by training on a single set of incoming flows. The resulting box plot using this set of flows is shown in Figure 4. For this scenario, the DRL approach is able to achieve similar statistics (median, average, and maximum utilization) as the theoretical benchmark solution. This result is remarkable, given that the agent is constrained to a set of candidate paths and solves the problem in a step-wise fashion.

It is worth noting that the agent implicitly learns the structure of future incoming flows, as these are identical across episodes during training. This enables the agent to optimize its decisions based on this implicit knowledge, explaining its ability to closely approximate the benchmark performance. However, the benchmark has complete knowledge of all flows upfront and can allocate them directly to minimize maximum utilization. This results in a significant portion of links being near the optimal utilization. In contrast, the DRL method includes some links with slightly higher load, as it does not have access to global information about all flows.

The histograms of the link load distributions underline this result. The proposed approach, shown in Figure 6, produces a distribution similar to the theoretical benchmark, shown in Figure 5. A promising future direction would be to generalize the agent to handle varying sets of flows. This could be achieved by employing a *curriculum learning* approach, where the agent is first trained on simpler flow scenarios and then gradually introduced to more diverse sets of flows. Such an approach would enable the agent to adapt to more dynamic and realistic environments while maintaining high performance.

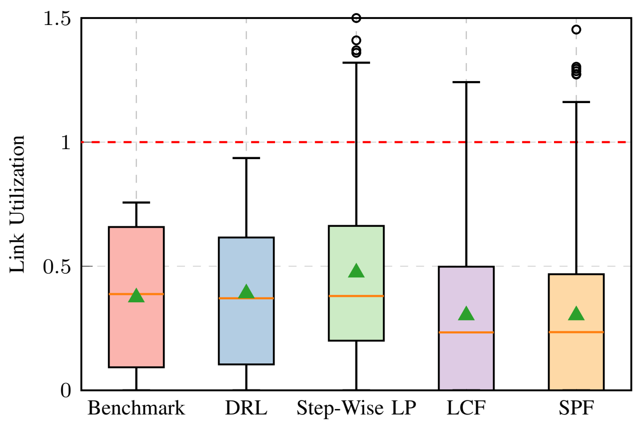

Fig. 4. Static scenario: box plot for deterministic set of 50 flows: comparing average (green triangle), median (orange line), minimum and maximum link utilization (caps). Benchmark is solution to MCFP using complete set, other approaches make step-wise decisions.

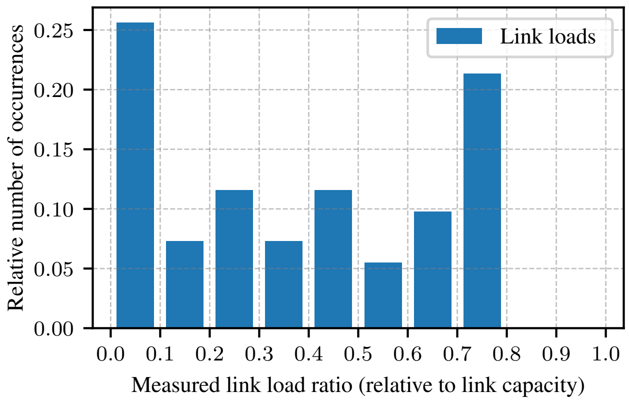

Fig. 5. Histogram of link loads for exemplary set of 50 flows for benchmark solution. Uses complete set at once to find optimal allocation.

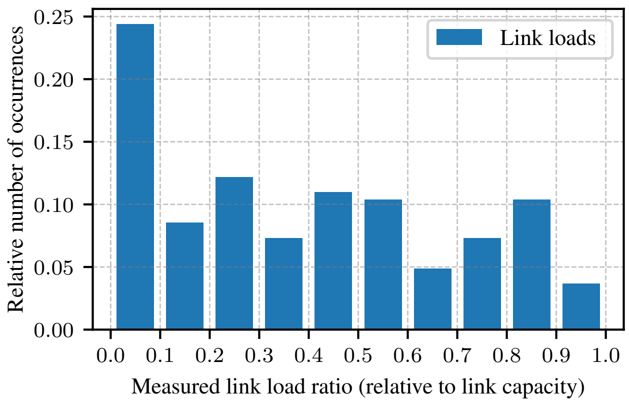

Fig. 6. Histogram of link loads for exemplary set of 50 flows for proposed DRL approach. Step-wise allocation of incoming flows.

2) Random Sets of Flows: The more general scenario consists of randomly sampled incoming flows. Thus, the agent must learn policies that are valid for a diverse and unpredictable set of potentially incoming flows. In this setting, the environment seen by the agent is inherently noisy, as the sampled flows vary significantly between episodes. Unlike the deterministic setting, the agent cannot rely on implicit learning of future flows but instead must base its actions on general traffic patterns dictated by the heat-map. This added complexity introduces greater uncertainty in the impact of the agent's actions.

Compared to the curriculum learning approach, where the agent is gradually exposed to increasingly complex scenarios, this approach confronts the agent with a diverse set of environments at once. While this may lead to slower convergence, it has the advantage of training the agent to be robust to variability in the environment, which is critical for real-world applications.

As we assume no prior knowledge of the flows, the theoretical benchmark is not applicable in this scenario. Moreover, for some flow sets, there may even be no feasible LP solution (too many constraints - overdetermined). Instead, we compare the results to the EMP method. To better evaluate the performance, we measure the total demand routed by each method until saturation is reached. For this scenario, we train a separate policy that terminates early whenever an incoming flow request cannot be routed due to link saturation and we use a simple reward function defined as

$$r_{t+1} = \begin{cases} d_t & \text{if the flow is successfully allocated,} \\ 0 & \text{otherwise.} \end{cases}$$

This reward function encourages the agent to maximize the number of successfully routed flow requests, thereby increasing the total demand routed and earning a higher cumulative reward.

We trained two policies: one utilizing the one-hot encoding strategy for the state space and the other using the histogram-based state space, as described in Section III-E. The training progression, including reward evolution for both policies, is depicted in Figure 8. The hyperparameters used for both policies are shown in Table II. For the histogram state space we used a binning size of $b = \frac{1}{10}$ which results into a state space of size $S_2 = 51$ (see (2)).

As the flow sets are randomized, we use Monte Carlo simulations to evaluate the proposed approach and compare it to EMP, LCF, and SPF. The results, based on $10 \cdot 10^3$ sampled episodes, are shown in Figure 7 and demonstrate that the agent learned viable policies. Each method operates on the same set of flows per episode until network capacities are saturated. The figure also compares SAC policies (DRL 1 with the histogram state space and DRL 2 with the one-hot encoding state space) to these baselines. The DRL approaches outperform the baselines by routing higher total demand, demonstrating the effectiveness of learned policies over heuristic methods.

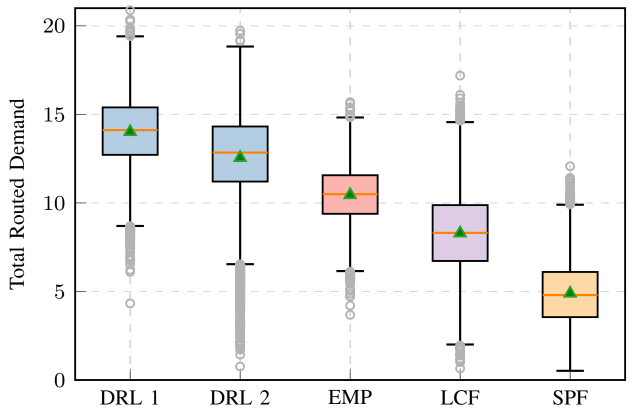

Fig. 7. Comparison of different routing methods for iteratively routing demands until overflow. The boxplots show the median (orange line), average (green triangle), and range (whiskers) of the total routed demand for each method.

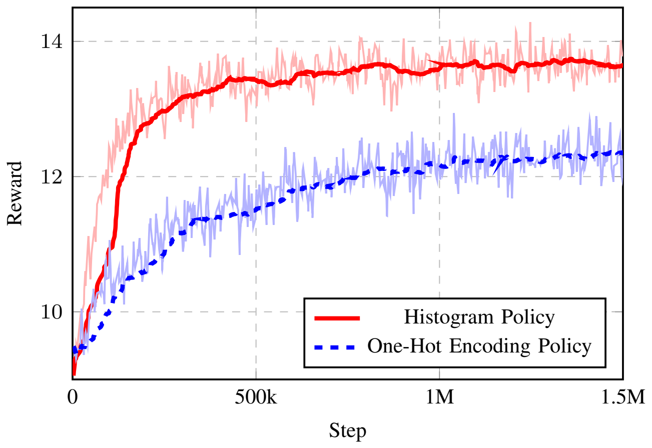

Fig. 8. Reward progression during training of the policies using the SAC algorithm. The plot shows both the raw reward data (lighter lines) and the smoothed reward data (solid lines) for each policy: the Histogram Policy (red) and the One-Hot Encoding Policy (blue). The smoothed data is obtained using a moving average with a window size of 50 steps, highlighting the overall trends in reward evolution over training steps and illustrating the learning behavior of the agent.

V. Discussion

A. *Strengths of Proposed Method*

A core strength of the DRL approach is its flexibility. With the proposed ML-based method, heat-maps and traffic patterns are learned directly from the resulting data. Decision-making is adjusted accordingly to anticipate bottlenecks and to maintain critical paths. As the set of candidate paths consists of suitable paths, there is an inherent rule-based baseline in the approach.

The complexity of inferring the models is also relatively low. The majority of the complexity is caused by computing the candidate paths. Relative to heuristics, no strong assumptions about network characteristics are necessary.

B. *Limitations*

There are two core limitations of the proposed approach: consistency of traffic patterns and the set of candidate paths.

Firstly, as the decision-making is step-wise, we have to rely on the reliability of the learned intuition. If the actual traffic requests differ from the expected distribution, i.e. the learned heat-map, performance degrades. In addition, we rely on the assumption that spreading traffic among the set of candidate paths achieves near-optimal solutions. Given the effectiveness of the proposed candidate path sets demonstrated in the deterministic scenario, this assumption seems to hold in the given scenarios. Additionally, increasing the action space to a larger set may improve performance.

For larger and more dynamic networks, pre-processing methods including Convolutional Neural Networks (CNNs) or Graph Neural Networks (GNNs) could be used for feature extraction instead of one-hot encodings. Given the rather small scale of the reference network, the considered encoding demonstrated similar results as a CNN-enabled encoding. As the histogram-based observations provide promising results, more compact representations may be sufficient.

C. Quality of Trained Policies

The learned policies should not simply correspond to a distribution of chosen actions. For the scenario based on a single deterministic set of flows, the considered actions are strongly distinct. There is no apparent underlying distribution. As the agent can converge towards a single suitable suite of actions, this is to be expected. However, for the second scenario, with sets of random flows, the agent may only learn a general policy distribution. Still, the observed policies were responsive to traffic patterns rather than relying on generic action distributions.

D. Implications for Real-World Deployment

Overall, once trained, the approach represents a relatively low-complexity solution to distribute incoming flows. It appears suitable for SDN-enabled SCNs, but may require more sophisticated models to manage real-world network dynamics and varying traffic patterns. The dynamic topology and changing link characteristics, e.g. latency, represent key implementation issues. Thus, the performance of the approach may vary for different scenarios. Nevertheless, given the generic nature of the method, there are interesting implications for further research.

VI. Conclusion

We have proposed and examined a SAC-based DRL approach to approximate solutions to the MCFP in SCNs with low complexity. By distributing traffic on sets of candidate paths, the scheme was able to outperform rule-based approaches in different scenarios. For a deterministic set of flows, the approach was able to learn an approximate solution to the MCFP. In a scenario with randomly sampled sets of flows, the learned policies enabled more flow allocations until link capacity was exceeded - thanks to learned knowledge of underlying traffic characteristics. The approach is designed to complement state-of-the-art routing protocol architectures, such as distributed SDN, for satellite networks by providing low complexity traffic engineering capacities. The robust performance of DRL in dynamic systems is also of interest for future applications in SCNs, especially for link outages and handovers. Further investigations in more diverse scenarios and in system-level simulators are intended to further demonstrate the viability for real-world deployment of DRL-based traffic engineering.

Acknowledgment

This work has been supported by the Space for 5G & 6G programme of the European Space Agency (ESA), activity code $5A.079$, https://connectivity.esa.int/projects/aicoms. Responsibility for the contents of this publication rests with the authors.

References

[1] D. Bhattacharjee, P. G. Madoery, A. U. Chaudhry, H. Yanikomeroglu, G. K. Kurt, P. Hu, K. Ahmed, and S. Martel, "On-Demand Routing in LEO Mega-Constellations With Dynamic Laser Inter-Satellite Links," *IEEE Transactions on Aerospace and Electronic Systems*, pp. 1–17, 2024.

[2] M. M. H. Roth, H. Brandt, and H. Bischl, "Distributed SDN-based Load-balanced Routing for Low Earth Orbit Satellite Constellation Networks," in *2022 11th Advanced Satellite Multimedia Systems Conference and the 17th Signal Processing for Space Communications Workshop (ASMS/SPSC)*, 2022, pp. 1–8.

[3] C. H. Papadimitriou and K. Steiglitz, *Combinatorial Optimization: Algorithms and Complexity*. Dover publ, 1998.

[4] K. Salimifard and S. Bigharaz, "The Multicommodity Network Flow Problem: State of the Art Classification, Applications, and Solution Methods," *Operational Research*, vol. 22, no. 1, pp. 1–47, 2022.

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

[6] P. Grislain, N. Pelissier, F. Lamothe, O. Hotescu, J. Lacan, E. Lochin, and J. Radzik, "Rethinking LEO Constellations Routing with the Unsplittable Multi-Commodity Flows Problem," in *2022 11th Advanced Satellite Multimedia Systems Conference and the 17th Signal Processing for Space Communications Workshop (ASMS/SPSC)*, 2022, pp. 1–8.

[7] K.-C. Tsai, L. Fan, L.-C. Wang, R. Lent, and Z. Han, "Multi-Commodity Flow Routing for Large-Scale LEO Satellite Networks Using Deep Reinforcement Learning," in *2022 IEEE Wireless Communications and Networking Conference (WCNC)*, 2022-04, pp. 626–631.

[8] M. M. H. Roth, A. Hegde, T. Delamotte, and A. Knopp, "Shaping Rewards, Shaping Routes: On Multi-Agent Deep Q-Networks for Routing in Satellite Constellation Networks," in *Proceedings of SPAICE 2024: The First Joint European Space Agency SPAICE Conference / IAA Conference on AI in and for Space*. Zenodo, 2024.

[9] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor.

[10] T. Haarnoja, A. Zhou, K. Hartikainen, G. Tucker, S. Ha, J. Tan, V. Kumar, H. Zhu, A. Gupta, P. Abbeel, and S. Levine. Soft Actor-Critic Algorithms and Applications.

[11] D. Eppstein, "Finding the k Shortest Paths," *SIAM Journal on Computing*, no. 28.2, p. 26, 1997.

[12] M. Towers, A. Kwiatkowski, J. Terry, J. U. Balis, G. De Cola, T. Deleu, M. Goulão, A. Kallinteris, M. Krimmel, A. KG, R. Perez-Vicente, A. Pierré, S. Schulhoff, J. J. Tai, H. Tan, and O. G. Younis. Gymnasium: A Standard Interface for Reinforcement Learning Environments. [Online]. Available: https://arxiv.org/abs/2407.17032

[13] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, "Stable-Baselines3: Reliable Reinforcement Learning Implementations," *Journal of Machine Learning Research*, vol. 22, no. 268, pp. 1–8, 2021. [Online]. Available: http://jmlr.org/papers/v22/20-1364.html

[14] A. A. Hagberg, D. A. Schult, and P. J. Swart, "Exploring Network Structure, Dynamics, and Function Using NetworkX," in *Python in Science Conference*, 2008, pp. 11–15. [Online]. Available: https://doi.curvenote.com/10.25080/TCWV9851