HELICON: Orchestrating low-latent & load-balanced Virtual Network Functions

Monchai Bunyakitanon, Xenofon Vasilakos, Reza Nejabati and Dimitra Simeonidou
Smart Internet Lab, Department of Electrical & Electronic Engineering,
University of Bristol, Bristol, Clifton BS8 1UB, UK
Email: name.surname@bristol.ac.uk

***Abstract*—HELICON is a novel hierarchical Reinforcement Learning (RL) approach for orchestrating the dynamic placement of Virtual Network Functions (VNFs) in Cloud and Edge 5G environments. It proves capable of addressing an NP-Hard decision-making problem with adopted RL while augmenting the current state of the art in orchestrators with a previously unexplored *lightweight distributed* and *hierarchical* RL approach. HELICON can run as a *fully autonomous* solution or complement orchestrators, thus *bridging a significant gap* in existing orchestrators, which generally lack intelligent and dynamic adaptation capabilities. Finally, our performance evaluation results over an actual 5G city testbed and use case validate that HELICON outperforms traditional policy-based Open Source MANO and other heuristic policies concerning single or multi-objective optimisation goals. What is more, HELICON's performance meets with that of node-specific custom supervised learning models, whereas it clearly outperforms supervised learning under dynamic conditions.**

***Index Terms*—Network function virtualization, Software defined networking, 5G mobile communication, Machine learning**

I. Introduction

Network softwarisation in the fifth generation of wireless networks (5G) is characterized by significant flexibility and agility as a result of adopting the concepts of Software-Defined Networking (SDN) and Network Function Virtualization (NFV). The former have enabled scalable vertical industry services with strict performance requirements that need to be addressed by MANagement and Orchestration (MANO) systems. Nonetheless, state of the art orchestrators such as ETSI Open Source MANO (OSM MANO) face challenges [1] out of which the NP-Hard [2]–[4] problem of *optimal* Virtual Network Function (VNF) *placement* remains essential.

Towards addressing this problem, we identify two main gaps: (i) orchestrators still lack Machine Learning (ML) *intelligence* with (ii) their majority [5] remaining rule- or heuristic-based, focusing exclusively on system-level resources after predefined policies. This approach neglects network dynamics and system-wide service-level performance objectives of both verticals and providers. To address these gaps, we propose a Hierarchical rEinforcement LearnIng approach for OrChestratiNg (`HELICON`) *low-latent* and *load-balanced* VNFs. Our contributions can be briefed as:

(1) A novel distributed hierarchical RL approach. `HELICON` can serve as a stand-alone online VNF placement solution as well as a module-based extension, hence *covering an intelligence gap* in current state of the art orchestrators. To the best of our knowledge, `HELICON` constitutes a unique Hierarchical Reinforcement Learning (HRL) effort of its kind.

(2) Tackling a difficult problem with a tunable and lightweight Q-Learning scheme. This work completes our prior effort [6] on optimizing either (i) *end-to-end (e2e) service delay* or (ii) load balancing among Compute Nodes (CNs). We extend our approach to a Multi-Objective Optimisation (MOO) solution capturing both objectives above based on a combination of Q-learning schemes with carefully designed reward functions and a *tunable* optimization priority. Notably, `HELICON` is *accurate* and *lightweight* by distributing predictions' load among global and local modules.

(3) Real testbed implementation and use case-driven validation. Most studies engage into simulation-based model evaluation over custom scenarios. Unlike that, we present *practical* experimental results upon a realistic 5G Smart City Safety (SCS) use case[1] conducted over a Bristol's 5G city testbed[2] assuming an e2e application video transcoding VNF. Our meticulous performance evaluation against *seven benchmark models* shows that `HELICON` performs significantly better compared to traditional OSM MANO policies and the other benchmarks under different Single-Objective Optimisation (SOO) and MOO scenarios.

II. Background and related work

VNF placement is highly challenging. It emerged with modern programmable networks adopting SDN and NFV principles. It is also related to the *Cloud-vs-Edge* dilemma in advanced wireless networks by considering the trade-off between exploiting a higher Cloud processing power against a reduced latency due to "Edge" network user proximity. Although the efficiency of VNF placement depends on processing power and network performance, most existing placement algorithms consider merely local resource availability for host selections. This provides no performance guarantees, especially for delay-sensitive VNFs. In addition, different VNF requests may have different objectives, possibly opposing one another such as in the case of clustering VNFs against load balancing. This context proves highly challenging with past literature like [2],

This work has received funding from the EU H2020 project 5GASP (project number 101016448).

[1] www.bristol.ac.uk/engineering/research/smart/5g-demonstrations/smart-city-safety/

[2] http://www.bristol.ac.uk/engineering/research/smart/5guk/

[7] discussing the *NP-hard* nature of the problem and its implications on programmable networks. Optimal host selection demands accurate VNF performance predictions *before* placement, dealing with hardware and software *heterogeneity*.

Reinforcement Learning (RL) addresses dynamic network optimization including VNF placement without any prior training via online learning [8]. Nevertheless, most works in the literature explore Supervised Learning (SL) techniques and particularly Artificial Neural Networks (ANNs), e.g. [9], with only a few exploring RL for VNF resource management. This is partially due to the non-trivial task of applying RL in networking. The greatest challenges regard designing the model itself, monitoring and feeding data to it, and the particular complexity [10] of MOO with RL.

Mao et al. [11] show the benefits of applying Deep RL *agents* to large-scale systems. Tesauro et al. [12] also demonstrate RL benefit over other model-based approaches derived from queuing theory. More recent studies focused either directly on RL or hybrid solutions. Chen et al. [13] optimize two Deep RL Agents (RLAs), namely, a Short flow RLA (sRLA) and a Long flow RLA (lRLA). Mijumbi et al. [14] also propose a multi-agent learning algorithm for virtual network resource management which significantly improves the acceptance ratio and the maximum number of accepted virtual network requests while compiling with Quality-of-Service (QoS). Vimal et al. [15] propose a multi-objective Ant Colony Optimization metaheuristic resource allocation algorithm.

Regarding our own work in the field, we have investigated adapted RL for local node performance SOO w.r.t. incoming placement requests [6]. The work there posses the basis for MOO in the current work. We have also investigated a rich range of different SL techniques for placing VNFs [16], concluding that local RL placement decisions are *portable*.

III. System Architecture

As portrayed in Figure 1, HELICON applies four different agents in order to improve VNF placement at the OSM MANO: (i) an application monitoring agent continuously tracking application performance, (ii) a node monitoring agent regarding CN resource utilization, (iii) a prediction agent predicting the VNF performance and (iv) a placement agent that places the VNF to the best location. There are two types of models depicted in the top part of Figure 1: (i) Local Reinforcement Learning (LRL) (ii) Global Reinforcement Learning (GRL). The LRL models are embedded in each CN and produce local predictions that are sent to the orchestration site where the GRL is deployed. The GRL suggests a node placement in response to incoming new VNF request. The placement suggestion can be made with either optimization scheme: (i) a *Single Objective Global Reinforcement Learning optimization scheme (SO-GRL))*; and (ii) a *Multi-Objective Global Reinforcement Learning optimization scheme (MO-GRL)* based on scalarizing 2 SO-GRLs.

Notice that the OSM node and the CNs are connected in one-to-many relationship. We represent this connection as a connected graph $G = (O, C)$, where O is the OSM node,

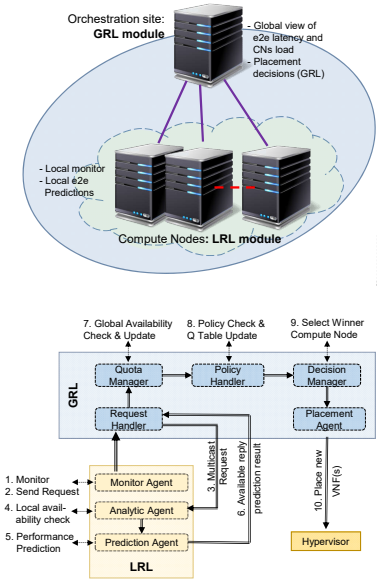


Fig. 1: High-level architecture and workflow. LRL agents integrate *fully* distributedly and run *fully* independently in CNs.

and $C = \{c_1, c_2, .., c_n\}$ is the set of CNs connected to OSM. CNs are also connected to end points like sensors or users in a one-to-many fashion. CNs only receive monitoring parameters from end devices. Each c has a resource capacity and status regarding the Central Processor Unit (CPU), memory (RAM), storage, and current load. We denote the CN resource capacity and status by $RS_c = (RS_c^{cpu}, RS_c^{mem}, RS_c^{disk}, RS_c^{load})$.

We use $A = \{a_{pl}, a_{mon}, a_{pre}\}$ as the set of embedded agents in the system where a_{pl} is installed in O and a_{mon}, a_{pre} are deployed in $\forall c \in C$. A set of VNF requests denoted by $F = \{f_1, f_2, .., f_n\}$, represents the series of request for new VNF(s). We implement a single use case in the test, which the resource demand is fix denoted in term of resources and status. An additional requirement (Accepted Delay: $AD = \{ad_1, ad_2, .., ad_n\}$) is applied when the end-to-end delay (t_{tot}) is considered for GRL. Therefore, the demand D_f is represented as $D_f = (D_f^{cpu}, D_f^{mem}, D_f^{disk}, D_f^{load}, D_f^{ad})$.

In addition, we state the resource constraint on CNs since multiple VNFs can be placed in a single node. Given n_c^f as the number of instances $f \in F$ instantiated on node $c \in C$, the constraint can be written as formula (1).

$$\forall c \in C : \sum_{f \in F} n_c^f C_f \leq C_c \tag{1}$$

A. Local prediction modules

Due to space restrictions, we briefly outline the local prediction models, with the details provided in [6]. We use two different LRLs to target two different objectives in the context of the SCS use case, namely: (i) e2e delay and (ii) CPU load balancing after VNF placement.

1) LRL for e2e delay: The State space (S) includes all possible T_{tot} predictions. The current state $s \in S$ is the basis for three possible next states s': (1) `Less than`: $PV < RV$; (2) `Equal`: $PV = RV$; and (3) `More than`: $PV > RV$, where Predicted Value (PV) refers to the predicted e2e delay by the model and Real Value (RV) to the actual e2e delay as measured after the placement. The actions set $A(s)$

contains actions a for increasing/decreasing T_{tot} by a step of 0.01. According to the SCS use case, the minimum-maximum value we can predict for T_{tot} is 0-2 s. Rewards $R_a(s, s')$ are *immediately* obtained *after executing an action* leading to s', hence reflecting the quality of the model prediction. Formula (2) gives immediate rewards en $[0, 1]$ using an υ[3] intensive band *tolerance margin* distance from a hyperplane.

$$R_a(s, s') = \begin{cases} 1 - |\frac{PV-RV}{\upsilon \cdot RV}|, & \text{if} |PV - RV| \leq \upsilon \cdot RV; \\ 0, & \text{otherwise}, \end{cases} \quad (2)$$

We adopt the actions strategies of *exploration* and *exploitation*, with the earlier identifying an initially unknown environment after random actions, and the latter taking actions after a *ϵ-greedy policy* (see Formula (3)) where $0 \leq n \leq 1$ is pseudo-randomly generated and compared against a decaying ϵ-greedy value with each consequent action. Finally, the state s changes to s' and a next prediction starts by creating a new Q-table with initial state $s \equiv s'$ and by considering a new tuple of the three-time samples $d_i = \{(T^i_{rcv}, T^i_p, T^i_{req})\}$.

$$a = \begin{cases} rand\ A(s), & \text{if n} < \epsilon\text{-greedy, exploration;} \\ arg\ \max_a Q(s, a), & \text{otherwise, exploitation} \end{cases} \quad (3)$$

2) LRL for load: System load refers to the number of CPU cores used by processes in execution or in a waiting state. The loads of VNF requests and candidate CN hosts are continuously monitored, allowing to calculate CNs' load after placement according to Formula (4), where $Load_{pred}$ is the Predicted Load (PL) after placing the VNF; $Load_{VNF}$ is the added load by the VNF; $Load_{node}$ is the candidate CN's current load; and $Core_{num}$ is CN's number of the CPU cores.

$$Load_{pred} = \frac{(Load_{VNF} + Load_{node})}{Core_{num}} \times 100\,. \quad (4)$$

B. SO-GRL

A Single Objective Global Reinforcement Learning optimization scheme (SO-GRL) is used to maintain Q-tables that used by MO-GRL. It takes input from LRLs and suggests a decision based on a single objective. The State space ($S(i)$) comprises all placement requests i en $l = \{0, 1, 2, ..., n\}$. $S(0)$ implies no VNF requests in the system, while $S(n)$ indicates that the system runs in full capacity. The actions set $A(s)$ contains actions a mapping the assignment of a CN to a number ($A(s) = \{1, 2, 3, .., m\}$). $A(s)$ is complete at the initial state $S(0)$ because all CNs are available. CNs with VNFs covering their full capacity keep leaving $A(s)$ in intermediate states until $A(s)$ becomes empty at $S(n)$. Note that if SO-GRL runs with MO-GRL, then it adopts the allocation decision by the central MO-GRL, calculates the reward and updates its Q-table. Otherwise, for stand-alone SO-GRL optimization, we apply the $\epsilon - greedy$ policy locally. SO-GRL reward (see Formula (5) subject to constraint (1)) for T_{tot} $R_t(s, s')$ represents the quality of the placement after T_{tot}. We define an Accepted Value (AV) as the maximum T_{tot} until which a VNF meets delay requirements.

$$R_t(s, s') = \begin{cases} \frac{1}{PV^{AV \times \omega_i}}, & \text{if } PV \leq AV; \\ min(\frac{1}{PV^{AV \times \omega_i}}, \frac{1}{PV - AV}), & \text{otherwise.} \end{cases} \quad (5)$$

[3] υ is used in several SL regressors such as Support Vector Regression (SVR). Our tests validated the use of $upsilon = 10\%$.

where ω_i is the fraction of AV of request i over the sum of the distinct AVs categories by other requests. The reward drops exponentially for larger PVs, with ω_i *tuning delay criticality* of request i relatively to the rest of the active requests' AVs.

Rewards for load predictions are based on standard deviation (std_{sel}), as measured after adding the PL of the selected node to the system, relative to average load ($ave(PL)$).

$$R_l(s, s') = \begin{cases} 1 - \frac{std_{sel}}{ave(PL)}, & \text{if } std_{sel} \leq ave(PL); \\ max(\frac{ave(std) - std_{sel}}{ave(std)}, 0), & \text{otherwise.} \end{cases} \quad (6)$$

The reward is designed to capture the impact of placements on *system-wide* load balance. Standard deviation is by default the most appropriate metric for assessing deviation from a balanced state where all nodes have the same load. Last, we divide std_{sel} by $ave(PL)$ to generate rewards exponentially.

C. MO-GRL

Global Reinforcement Learning (GRL) is a high level model that makes the final VNF placement decision based on the Q-tables of both SO-GRLs. MO-GRL uses the same State and Action as the SO-GRL (see Section III-B). For the action selection, it applies the ϵ-greedy strategy as described in the formula (3). MO-GRL does not have the reward function. It gives the reward $R_s(s, s')$ from the scalarization of the two Q-values from SO-GRLs, which are normalized through dividing the given reward by the maximum reward of the same state, using the formula (7) and subjected to constraint (1).

$$R_s(s, s') = (\omega_t \cdot Q_t(s_t, a_t)) + (\omega_l \cdot Q_l(s_t, a_t)) \quad (7)$$

where ω_t and ω_l are the coefficients of weight for LRL for T_{tot} and load balance, $Q_t(s_t, a_t)$ and $Q_l(s_t, a_t)$ are normalized Q-value for LRL for T_{tot} and load balance respectively.

IV. Experimental plan

The details of our city-wide testbed are described in our prior works of [6], [16]. Regarding our evaluation plan, it involves an analytical performance comparison against *7 benchmark models*: **(1) OSM:** We use an orchestration scheme traditionally used by OSM MANO notated as (i) ``OSM''. In short, OSM considers *only* physical resources in accordance with the NOVA filter scheduler[4]. It is a *non*-ML benchmark that is based on two strategies: "*filtering*" and "*weighting*". Filters are sets of rules defining the resources and capabilities of a CN for hosting a VNF, whereas the weighting strategy applies weights to all filters to define their influence on decisions. **(2)-(4) HRT:** We implement three *heuristic* placement models: (v) HRT(Delay), (vi) HRT(Load), and (vii) HRT (Delay; Load). The former apply the same ranking approach as Ben based on an *intelligent RL*-prediction approach known as "Latency-aware" [17], and the current load. **(5)-(7) Ben:** We use three intelligent heuristic VNF placement models with prediction modules during MOO experiments. The first two target a single objective, namely: (ii) Ben(Delay) tries to minimize e2e service delay based on node-local ANN models that are designed, trained and deployed for predicting T_{tot} in their local nodes; (iii) Ben(Load) selects the CN that

[4] docs.openstack.org/ocata/config-reference/compute/schedulers.html

greedily minimizes load imbalances between CNs for consequent requests; while (iv) `Ben(Delay; Load)` applies a ranking system as a Mixed-integer linear programming approach, combining the former two single objective benchmarks. Moreover, all models are subject to the constraint (1). Regarding `Ben(Delay; Load)`, all local ANN-predicted T_{tot} values are sorted and assigned with a ranking integer i_t in ascending order. Likewise, all Load Balancing Score (LBS) values are ranked with a score value i_l in ascending order from best (minimum) to the worst (maximum) CPU load imbalance. Then, we linearly scalarize the sum of the corresponding ranking numbers $i = i_t + i_l$ and select the node with the minimum scalarized value.

Regarding ***evaluation metrics***, we adopt: (i) $\mathtt{T_{tot}}$ is the system-wide *e2e delay* after all VNF requests are instantiated at hosts. It captures the overall delay from the camera to the selected CNs and from there to the final data consumption nodes. A good quality of placement should yield $500\,ms \leq T_{tot} \leq 700\,ms$ values according to SCS steady-state performance. (ii) `LBS` is the system-wide standard deviation of CPU usage after selecting a hosting CN s. A perfect *system-wide load balance* should yield $LBS = 0$. (iii) `NS` is the average Number of Selections (NS) of a CN during a requests round. It measures the placement frequency distribution among CNs.

A. Evaluation scenarios

1) Scenario 1: Single Objective Optimization: This scenario is designed to evaluate the performance of SO-GRL for delay and Load balance against `OSM` and `HRT` benchmarks under varying conditions. Each VNF requires 2 CPU cores 2 GB hard disk and 2 GB of RAM. Our testbed has a total capacity of 12 VNFs (CN1:4; CN2:4; CN3:2; CN4:2) as specified in Sec. IV. Each incoming VNF request is assigned randomly with one AV of either 700, 900, and 1100 ms, thus requirements range from a very strict to an easier AV. Last, we submit request batches of 6 and 18 VNFs to evaluate SO-GRL when request batches are either within or over system capacity.

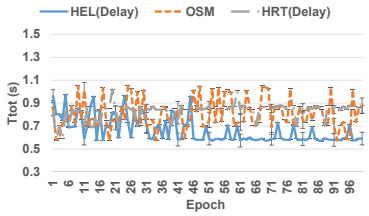
2) Scenario 2: Multi-Objective Optimization: We assess the scalarization quality of the two SO-GRL Q-tables, setting three different ratios between ω_t and ω_l as (i) `HEL(Delay 75; Load 25)` (emphasizing on minimizing T_{tot}); (ii) `HEL(Delay; Load)` (equal 0.50:0.50 importance); and (iii) `HEL(Delay 25; Load 75)` (emphasizing on minimizing LBS). VNF requests have the most strict AV (700 ms).

V. Performance Evaluation

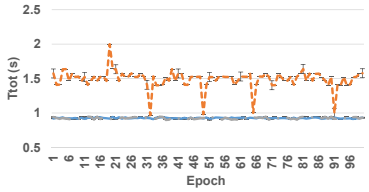
Models are exhaustively trained after 100 iterations, with initial Q-learning parameters: $<\alpha$=0.1; γ=0.9; $\epsilon-greedy$= 0.5; decay=0.97$>$. We present mean values and 95-percentile confidence intervals. Time-transient results exhibit performance convergence, whereas aggregate results mean performances over 2000 repeats in steady-state after epoch 80.

A. Single Objective Optimization

1) Delay: Graph 2(a) demonstrates the system-wide T_{tot} for 6 VNF batch requests. T_{tot} converges to 0.6 s after a first training and learning period until epochs 46-50, `HELICON`, and remains mostly ∼20 s less compared to `OSM` and HRT. For 18 VNF requests (Graph 2(b)), `HELICON` and HRT



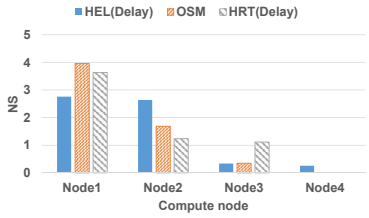
(a) 6 VNF batch requests



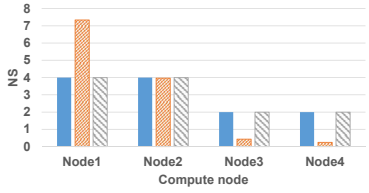
(b) 18 VNF batch requests

Fig. 2: Transient T_{tot} performance comparison.

outperform `OSM`, demonstrating a consistent 0.9 s T_{tot} against ∼1.5 s by `OSM`. Only 12 out of the 18 requested placements occur due to system capacity constrains, explaining why the performances of `HELICON` and HRT meet. HRT may be underlined by an already trained CN-local RL model, yet clearly `HELICON` optimizes placement decisions better when dealing with within-capacity constraints as we explain with



(a) 6 VNF batch requests



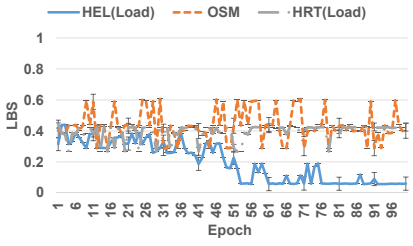
(b) 18 VNF batch requests

Fig. 3: Average Node Selection for optimizing T_{tot}.

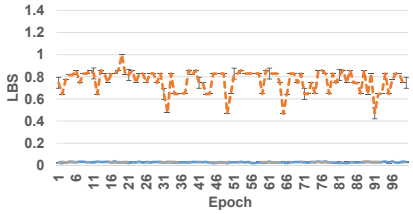
the help of Graphs 3(a) and 3(b). The latter graphs reveal an identical allocation behavior of `HELICON` and HRT regarding 18 requests and a slightly different one for the case of 6 requests. For `HELICON`, CN 1 and CN 2 are the most frequently selected ones in both graphs. Node selection numbers are evenly distributed between CN 1 and CN 2, as well as between CN 3 and CN 4 because these pairs of nodes have the similar resource capabilities leading to likewise T_{tot} performances.

Last, HRT's selection of the inferior performance node CN 3 for 6 VNFs in Graph 3(a) is significant, explaining why HELICON outperforms HRT's delay after reaching a steady state in Graph 2(a).

2) Load: Figure 4 shows transient load performance in terms of LBS for the cases of 6 and 18 VNFs requests. Notice the similarities to the performance patterns previously observed for delay in Figure 2. For 6 VNF batch requests, HELICON's LBS is ∼0.3 until epoch 46, and then drops and converges to ∼0.1 during the remainder epochs. Unlike that, LBS remains around 0.4 for both OSM and HRT(Load) with a more intense fluctuation in the case of OSM) during all epochs. Regarding 18 VNF batch requests, HELICON



(a) 6 VNF batch requests



(b) 18 VNF batch requests

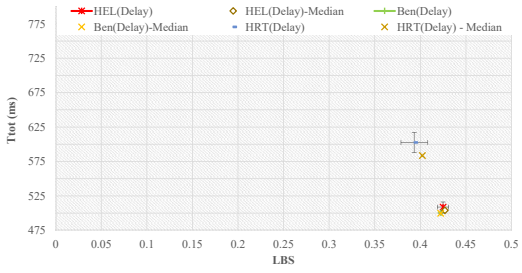
Fig. 4: Transient Load performance comparison.

LBS transient values are much lower than OSM's 0.4-0.8. This reveals the performance merits of adding intelligence to placements rather than using traditional orchestration methods. Given that requests exceed system capacity, resource utilization is 100% and a non-zero LBS denotes that *OSM fails to utilize* all available resources. Unlike that, and likewise to results in Graph 2(b), HELICON and HRT converge both to optimal LBS. Due to intelligence, the system is not only highly balanced, but also utilizes its full capacity.

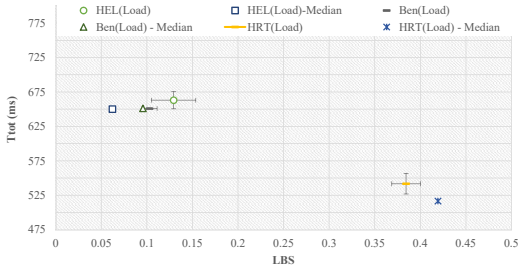
B. Multi-Objective Optimization

Apart from mean performances in steady-state, the Graphs of Figures and 6 show also median values for the sake of a more accessible analysis in dense parts of the Graphs. Note that we omit OSM results. Besides its proven poor performance in Scenario 1, OSM's traditional placement approach is inherently incapable of targeting more than one objectives.

1) Delay-only optimization: Specifically, Graph 5(a) shows that HELICON's mean T_{tot} HEL(Delay) exhibits the same performance as Ben(Delay) (∼500 ms). An essential conclusion is that HELICON manages to have at least the same performance as an individually trained and deployed ANN model per CN. This is *highly significant*: it denotes that



(a) Delay-only objective.



(b) Load-only objective.

Fig. 5: HELICON vs. benchmark schemes. Notice the focused scale on both axes (x: load; y: delay) to accommodate analysis.

HELICON *can meet the performance of a custom ANN model* even under static networking conditions despite the previously identified [16] merits of SL models under such conditions.The former comes as a result of considering system-wide performance parameters, which in this case proves to be *at least as good as* optimizing individualized nodes' performance parameters. Also, HEL(Delay) outperforms both mean and median HRT(Delay) by 16.6%/15% (100 ms/75 ms), hence demonstrating both a better and more stable performance against outliers. Last, all models exhibit almost identically poor LBS performances compared to Graph 5(b), as they all ignore load balance in their decisions.

2) Load balancing-only optimization: HEL(Load) in Graph 5(b) seems to slightly underperform compared to Ben(Load). However, as HEL(Load)--Median shows, this is only due to (significant) outliers. In fact, HEL(Load)--Median has LBS = 0.062, which is 38% less than for Ben(Load)--Median that accounts for ∼0.1. Regarding T_{tot} on the y-axis, both models exhibit the same performance (650 ms). In addition, we observe that HEL(Load) outperforms HRT(Load) significantly with an LBS that is almost x3.3 less than HRT's. Nevertheless, inadequate decisions w.r.t. load seem to boost delay performance for HRT revealing a trade-off between the two optimization goals to be further discussed next with the help of the results in Figure 6.

3) Joint Delay & Load balancing optimisation: According to Figure 6, for the case of equal importance between delay and Load, HEL(Delay; Load) performs better against BEN(Delay; Load) regarding load in trade off a higher delay cost. HEL(Delay; Load) yields $< T_{tot}, LBS >$=<602, 0.275> whereas BEN(Delay; Load) $< T_{tot}, LBS >$=<532, 0.335>. HRT(Delay; Load), on the other, exhibits a much worse performance

regarding both metrics. The cases of *non-equal importance* between Delay and Load (denoted as `HEL(Delay)`, `HEL(Delay 75; Load 25)`, `HEL(Delay 25; Load 75)` and `HEL(Load)`) have no direct equivalent benchmarks to compare, and we present them to study the impact of tuning the balance between T_{tot} and LBS. Balance changes affect performance measurements in an anticipated way. Observing all result points from left to right in the graph, which map to `HELICON` versions targeting from 100% Load and 0% Delay up to gradually 100% Delay and 0% Load with a 25% step, the improvement (decrease) of T_{tot} becomes *increasingly crucial* from ∼21ms to eventually 53ms. As also expected, Load performance becomes gradually worse (LBS increases).

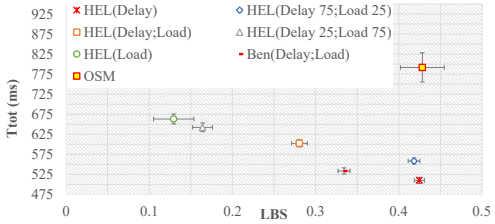


Fig. 6: `HELICON` against all benchmarks.

4) Dynamic network conditions: Following Graph 5(a) conclusion that `HELICON` can meet with the performance of custom trained ANN-based `Ben` models under static networking conditions, the results of Figure 7 study and reveal the merits of `HELICON` under more *realistically* dynamic 5G network conditions where CNs may arbitrarily join or depart. The results show that `HELICON` outperforms `Ben`

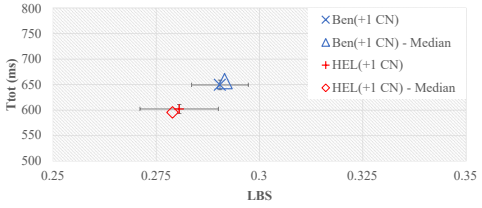


Fig. 7: Performance against the ANN-based `Ben(Delay; Load)` under dynamic conditions. `+1 CN` indicates adding a CN while in operation.

w.r.t. both delay and load balancing. Specifically, `Ben(±1% CN)` yields +10.4% delay (T_{tot}=657) compared to `HEL(±1% CN)` (T_{tot}=595), and +4.6% (LBS=0.292) as opposed to `HEL(±1% CN)` (LBS=0.279). Notice that median values almost identify with means, denoting the absence of outliers. The results align with [16], as statically trained and deployed local ANN models *cannot* adapt to new conditions. However, with `HELICON`, both the node-local and the top-hierarchy models adapt online to new conditions.

VI. Conclusions and Future Work

`HELICON` is a novel RL approach for orchestrating the dynamic placement of VNFs. It can serve as a fully autonomous online solution or a distributed module-based extension to current orchestrators. We tackle an NP-hard problem with a tunable, accurate and lightweight hierarchical scheme based on a combination of two Q-learning model schemes running at a global or local scale. Our practical testbed evaluation shows that `HELICON` outperforms `OSM` policy-based, other ML and heuristic-based orchestration regarding both SOO and MOO. For future work, we plan to explore three or more objectives, and to investigate approaches besides scalarization [18]. Finally, we will study the impact of non-uniform resource requirements per issued VNF request.

References

[1] R. Ranjan, B. Benatallah, S. Dustdar, and M. P. Papazoglou, "Cloud Resource Orchestration Programming: Overview, Issues, and Directions," *IEEE Internet Computing*, vol. 19, no. 5, pp. 46–56, 2015.

[2] S. Q. Zhang *et al.*, "Joint NFV placement and routing for multicast service on SDN," in *2016 IEEE/IFIP IEEE Network Operations and Management Symposium (NOMS)*, 2016, pp. 333–341.

[3] F. Carpio, S. Dhahri, and A. Jukan, "VNF placement with replication for load balancing in NFV networks," in *IEEE International Conference on Communications, ICC 2017, Paris, France, May 21-25*, 2017, pp. 1–6.

[4] D. Li *et al.*, "Virtual Network Function Placement with Function Decomposition for Virtual Network Slice," in *IEEE Conference on Standards for Communications and Networking, CSCN 2018, Paris, France, October 29-31, 2018*. IEEE, 2018, pp. 1–4.

[5] M. F. Bari *et al.*, "Orchestrating Virtualized Network Functions," *IEEE Trans. Network and Service Management*, vol. 13, pp. 725–739, 2016.

[6] X. Vasilakos, M. Bunyakitanon, R. Nejabati, and D. Simeonidou, "Towards Low-latent &amp Load-balanced VNF Placement with Hierarchical Reinforcement Learning," *IEEE International Mediterranean Conference on Communications and Networking*, 2021.

[7] M. Xia *et al.*, "Network function placement for nfv chaining in packet/optical datacenters," *Journal of Lightwave Technology*, vol. 33, no. 8, pp. 1565–1570, 2015.

[8] R. Boutaba *et al.*, "A comprehensive survey on machine learning for networking: evolution, applications and research opportunities," *J. Internet Serv. Appl.*, vol. 9, no. 1, pp. 16:1–16:99, 2018. [Online]. Available: https://doi.org/10.1186/s13174-018-0087-2

[9] C. W. Ahn and R. S. Ramakrishna, "Qos provisioning dynamic connection-admission control for multimedia wireless networks using a hopfield neural network," *IEEE Transactions on Vehicular Technology*, vol. 53, no. 1, pp. 106–117, Jan 2004.

[10] C. Liu, X. Xu, and D. Hu, "Multiobjective Reinforcement Learning: A Comprehensive Overview," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 45, no. 3, pp. 385–398, 2014.

[11] H. Mao *et al.*, "Resource Management with Deep Reinforcement Learning," in *15th ACM Workshop on Hot Topics in Networks - HotNets '16*. Atlanta, GA, USA: ACM Press, 2016, pp. 50–56.

[12] G. Tesauro *et al.*, "Online resource allocation using decompositional reinforcement learning," in *AAAI*, vol. 5, 2005, pp. 886–891.

[13] L. Chen *et al.*, "AuTO: Scaling Deep Reinforcement Learning for Datacenter-scale Automatic Traffic Optimization," in *2018 Conference of the ACM Special Interest Group on Data Communication*, ser. SIGCOMM '18. New York, NY, USA: ACM, 2018, pp. 191–205.

[14] R. Mijumbi *et al.*, "Design and evaluation of learning algorithms for dynamic resource management in virtual networks," in *IEEE Network Operations and Management Symposium (NOMS)*, May 2014, pp. 1–9.

[15] S. Vimal *et al.*, "Enhanced resource allocation in mobile edge computing using reinforcement learning based moaco algorithm for iiot," *Computer Communications*, vol. 151, pp. 355 – 364, 2020.

[16] M. Bunyakitanon, X. Vasilakos, R. Nejabati, and D. Simeonidou, "End-to-End Performance-based Autonomous VNF Placement with adopted Reinforcement Learning," *IEEE Transactions on Cognitive Communications and Networking*, pp. 1–1, 2020.

[17] M. Bunyakitanon, A. P. da Silva, X. Vasilakos, R. Nejabati, and D. Simeonidou, "Auto-3p: An autonomous vnf performance prediction & placement framework based on machine learning," *Computer Networks*, vol. 181, p. 107433, 2020.

[18] R. Yang *et al.*, "A Generalized Algorithm for Multi-Objective Reinforcement Learning and Policy Adaptation," in *NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, 2019, pp. 14 610–14 621.