A Reinforcement Learning Approach for Placement of Stateful Virtualized Network Functions

Godfrey Kibalya*, Joan Serrat*, Juan-Luis Gorricho*, Doreen Gift Bujjingo†, Jonathan Sserugunda†,Peiying Zhang ‡
*Dept.Network Engineering, Universitat Politecnica de Catalunya,Barcelona, Spain
†Dept. Electrical and Computer Engineering, Makerere University, Kampala, Uganda
‡College of Computer Science and Technology, China University of Petroleum (East China), Qingdao 266580, P.R. China
E-mails: {Godfrey.mirondo.Kibalya@upc.edu, serrat@tsc.upc.edu, juanluis@entel.upc.edu,
doreenserene04@gmail.com, serugthan@gmail.com, zhangpeiying@upc.edu.cn}

***Abstract*—Network softwarization increases network flexibility by supporting the implementation of network functions such as firewalls as software modules. However, this creates new concerns on service reliability due to failures at both software and hardware level. The survivability of critical applications is commonly assured by deploying stand-by Virtual Network Functions (VNFs) to which the service is migrated upon failure of the primary VNFs. However, it is challenging to identify the optimal Data Centers (DCs) for hosting the active and stand-by VNF instances, not only to minimize their placement cost, but also the cost of a continuous state transfer between active and stand-by instances, since a number of VNFs are stateful. This paper proposes a reinforcement learning (RL) approach for the placement of stateful VNFs that considers a joint reservation of primary and backup resources with the objective of minimizing the overall placement cost. Simulation results show that the proposed algorithm is optimized in terms of both acceptance ratio and cost, resulting in up to 27% and 30% improvements in terms of accepted requests and placement cost compared to a state-of-the art algorithm.**

***Index Terms*—Resilient Embedding, Reinforcement Learning, Stateful VNF, Service Function Chainning.**

I. Introduction

Network Virtualization enables the implementation of network functions as software modules running on general purpose hardware, resulting in efficient resource utilization since softwarized functions can be dynamically scaled according to real-time resource requirements [1]–[3]. However, softwarization increases the prospects of service failure due to possible failures at both hardware and software level, including software faults or misconfigurations, among other reasons [4]–[6].

The service survivability problem has been addressed in several works, as in [1], [7]–[14]. However, these and most of the existing works only consider stateless VNFs, where the resource consumption and placement of backup instances is independent of the active instances location. However, stateful VNFs generate states during traffic processing that ought to be transferred to stand-by instances in order to achieve a seamless traffic rerouting upon failure [15]. This may result in an excessive consumption of bandwidth resources, or even link congestions, since such updates need to be performed continuously, as long as the active instances are running [16]. Therefore, besides the amount of resources assigned to active and stand-by instances, the stateful placement problem needs to jointly consider the distance in between active and standby instances.

The stateful VNF placement problem has been addressed in [15], [17] and [16]. In [15] and [17] a Data Center (DC) with the highest rank, in terms of computational and bandwidth resources, is favored to host an active instance, while the DC that results in the least resource cost to the above DC is chosen to host the stand-by instance. Machine learning has emerged as a promising approach for problems in which the placement objective is jointly influenced by multiple attributes (e.g. monetary cost, load balance or energy consumption) due to its ability to infer the influence of the different features onto the decision objective. In this regard, the work in [16] adopts a reinforcement learning approach for the online fault-tolerant placement of VNF chains. However, the action space in that work grows as $\frac{N_{DC}!}{(N_{DC}-2)!}$ where N_{DC} is the number of DCs. This greatly impacts the algorithm convergence, hence, its performance for a manageable number of training episodes. Different from these works: i) we use the deployment cost as our placement objective, instead of the acceptance ratio which may bias the placement algorithm towards admitting requests of low resource requirements, resulting in less revenue; ii) we tame the action space size by sequentially selecting the host for the active and stand-by instances, but with the two selection steps been coordinated; iii) we use a convolutional neural network architecture due to its fast training speed. This way, in this paper we first present a formal formulation of the fault-tolerant stateful VNF placement problem, involving the joint reservation of resources for both active and stand-by instances. Then, a policy gradient based algorithm for the SFC placement is proposed, and its performance is evaluated through extensive simulations.

The rest of the paper is organized as follows: Section II

This work has received funding from the European Union's Horizon 2020 re-search and innovation programme under grant agreement No 777067 (NECOS pro-ject). This work is also funded by the national project TEC2015-71329-C2-2-R (MINECO/FEDER).

978-3-903176-32-4

presents the network model and problem description. The proposed RL based algorithm for stateful VNF placement is given in section III, and its performance evaluation is presented in section IV. The paper is concluded in section V.

II. Service, Network model and Problem description

A. SFC Request

We model each received request $r \in R$, as a tuple $r^{SFC} = < G_v^r, C_{dem}^r, Bwt_{dem}^r, Bws_{dem}^r, Del_{sd}^r, \tau_s^r, \tau_d^r, \tau_f^r >$, where G_v is the SFC connectivity graph. C_{dem}^r, Bwt_{dem}^r and Bws_{dem}^r denote the request requirements in terms of computing resources, transmission rate and state-update rate. The parameters Del_{sd}^r, τ_s^r, τ_d^r, τ_f^r denote the end-to-end latency requirement, the source node, the terminal/destination node and the request life-time.

B. Substrate network

This is modeled as an undirected graph G=(V $\cup$ Dc, E) where V, Dc, and E denote: the set of switches, DCs attached to V, and edges [15], [16]. Each $DC_j \in D_c$ is characterized by a computing resource capacity $C_{max}^{DC^j}$ and a unit rate processing cost $\rho_{DC^j}^p$. Likewise, each interconnecting link $e \in E$ is characterized by a bandwidth resource capacity B_{max}^e, a delay del^e, and a per unit rate transmission cost ρ_e^t.

C. Fault-tolerant VNF Placement problem description

The problem addressed in this work involves the selection of feasible DCs to host the active and stand-by instances with the goal of minimizing the request placement cost. This cost involves the processing cost at the selected DCs and the transmission cost at the edges used for both traffic and state-update flows. Therefore, the SFC placement objective minimizes the average placement cost per request, mathematically formulated as:

$$Minimize \quad \frac{1}{|R|} \sum_{r \in R} c(Gv)^r \tag{1}$$

where $c(Gv)^r$ is the placement cost for request $r \in R$. To formulate $c(Gv)^r$, we let $x_{DC^j}^r \in \{0,1\}$ and $\gamma_{DC^j}^r \in \{0,1\}$ denote binary variables equal to one if the active instance of request $r \in R$ is placed on $DC^j \in Dc$ and if its stand-by instance is placed on $DC^j \in Dc$, zero otherwise. Similarly, we let $\sigma_e^{r,act} \in \{0,1\}$, $\sigma_e^{r,bk} \in \{0,1\}$ and $\sigma_e^{r,su} \in \{0,1\}$ denote binary variables equal to 1 if the substrate edge $e \in E$ is used in the active, backup or state update paths, zero otherwise. Then, the cost $c(Gv)^r$ is computed as:

$$c(Gv)^r = C_{prcs} + C_{trans} \tag{2}$$

where the processing cost C_{prcs} is computed as:

$$C_{prcs} = \sum_{DC^j \in Dc} x_{DC^j}^r \rho_{DC^j}^p C_{dem}^r + \sum_{DC^j \in Dc} \gamma_{DC^j}^r \rho_{DC^j}^p C_{dem}^r \tag{3}$$

where the first and second terms in Eqn. 3 come from the cost of the active and stand-by instances. The transmission cost, C_{trans}, is computed as shown in Eqn. 4 with the first, second and third terms corresponding to the active instance, backup instance and state update costs.

$$C_{trans} = \sum_{e \in E} \sigma_e^{r,act} \rho_e^t Bwt_{dem}^r + \sum_{e \in E} \sigma_e^{r,bk} \rho_e^t Bwt_{dem}^r + \sum_{e \in E} \sigma_e^{r,su} \rho_e^t Bws_{dem}^r \tag{4}$$

In solving Eqn. 1, we adhere to the following constraints:

$$\gamma_{DC^j}^r \times x_{DC^j}^r = 0 \ \ \forall DC^j \in Dc, \forall r \in R \tag{5}$$

$$\sum_{r \in R} x_{DC^j}^r C_{dem}^r + \sum_{r \in R} \gamma_{DC^j}^r C_{dem}^r \leq C_{max}^{DC^j} \ \ \forall DC^j \in Dc \tag{6}$$

$$\sum_{r \in R} \sigma_e^{r,act} Bwt_{dem}^r + \sum_{r \in R} \sigma_e^{r,bk} Bwt_{dem}^r + \sum_{r \in R} \sigma_e^{r,su} Bws_{dem}^r \leq B_{max}^e \ \ \forall e \in E \tag{7}$$

$$\sum_{DC^j \in Dc} \gamma_{DC^j}^r \geq K \ \ \forall r \in R \tag{8}$$

$$\sum_{e \in E} \sigma_e^{r,act} del^e + \sum_{DC^j \in Dc} x_{DC^j} del_{DC^j}^r \leq Del_{sd}^r \ \ \forall r \in R \tag{9}$$

$$x_{DC^j}^r, \gamma_{DC^j}^r, \sigma_e^{r,act}, \sigma_e^{r,bk}, \sigma_e^{r,su}, \in \{0,1\} \tag{10}$$

From Eqn. 5, the active and stand-by instances of a request should be placed on different DCs. Eqns. 6 and 7 are capacity constraints. From Eqn. 8, the number of stand-by instances should not be less than a desired value K. Eqns. 9 and 10, respectively, enforce delay and domain constraints.

The above problem is a common NP-hard problem of unfeasible resolution for delay-sensitive applications when using exact solvers. This motivates the proposed approach which is able to obtain near-optimal solutions in acceptable run-time.

III. Proposed RL based VNF placement algorithm

In this section the proposed algorithm is explained, focusing on the adopted MDP model and its neural network architecture.

1) MDP model: This is defined as a 4-tuple (S, A, P, R) where S, A, P, R denote: the system state, the action space, the state transition probability and the reward signal respectively, as discussed below:

State : We formulate the system state at a given time as a $|Dc| \times N$ feature matrix, where Dc is the set of all DCs and N is the number of features associated with each DC. For each request $r \in R$ to be placed, the features associated with each possible $DC^j \in Dc$ in the network are: the processing cost per unit rate, the residual computational resources, the number of edges between source and terminal nodes, and the delay and available bandwidth along the source to terminal path traversing DC^j.

Action space : This is made up of all the available Dcs from which the host of the active and stand-by instances can be selected. The policy to be learned first selects the DC_{act}, then the DC_{std}, where the selection of the DC_{std} is influenced

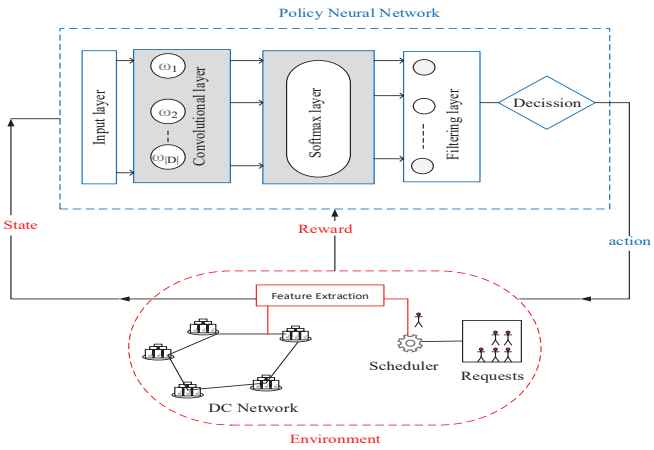


Fig. 1. Policy Neural Network Architecture and Interaction

by the selected DC_{act}. Therefore, the action set for a request is an ordered sequence [DC_{act}, DC_{std}].

Reward : We align the reward definition with equation 1 and formulate the normalized reward as follows:

$$Reward = \frac{ideal\_cost}{actual\_cost} \tag{11}$$

where the $ideal\_cost$ refers to the least possible cost that can be incurred in placing the request. This corresponds to the case in which both the active and stand-by instances are placed on DCs that are one hop away from each other. Moreover, the source and destination nodes are one hop away from the selected DC_{act} and DC_{std}. The term $actual\_cost$ corresponds to the actual incurred cost as given in Eqn. 2.

2) Policy Neural Network architecture: Fig. 1 shows the adopted architecture for the policy neural network. For a scheduled request, the feature extraction block uses the current substrate network state and the request requirements to output a state matrix which serves as input to the policy neural network. The neural network architecture consists of 4 layers: the input, convolutional, softmax and filtering layers. The convolutional layer performs a convolution between the input state matrix and the learnable weights of the layer. The softmax layer transforms the convolutional layer output into a vector of probabilities of each DC to be selected for instantiating the VNF. The filtering layer prunes out unfeasible DCs, after which, the DC with the highest probability is selected as the host for the active/stand-by instance, except for the training phase, where a trade-off between exploitation and exploration must be carried out.

3) Policy network training: The neural network is trained using 1000 off-line requests for each training epoch, with request parameters specified in section IV-A, for a total of 100 epochs. For each SFC to be placed, a feature matrix to select the DC_{act} is generated and fed into the policy network. Since the neural network parameters are initially randomly assigned, we perform a trade off between exploration and exploitation. If the selection of a DC_{act} is unsuccessful, the request is rejected; otherwise, for the selected DC_{act}, the gradients of the policy neural network are computed and stacked. Then, a new feature matrix to select a DC_{std} is generated taking into account the updated resources and the selected DC_{act}. If the selection of a DC_{std} is successful, for this DC_{std}, the gradients of the neural network are computed and stacked together with the previous ones for DC_{act}. The resulting reward signal is computed according to Eqn. 11; otherwise, the stacked gradients are deleted and the request is rejected. The definitive gradient of a successful request is computed as:

$$g := \alpha.r.g_s \tag{12}$$

where α, r, and g_s denote the learning rate, reward and stacked gradients. Gradients from different requests are stacked into a common buffer until reaching a given batch size. Then, all gradients previously stacked are jointly applied to the model, and the common buffer is emptied.

IV. Performance Evaluation

The proposed RL and the benchmark algorithms are compared in terms of: i) average acceptance ratio (AR) computed as the fraction of accepted requests to the total requests in the system; ii) active cost which is the cost incurred in placing the active instances; iii) stand-by cost which is the cost incurred for reserving backup resources; iv) state update cost which is the cost for transferring the update information; v) aggregate cost which is the sum of active, state update and stand-by costs.

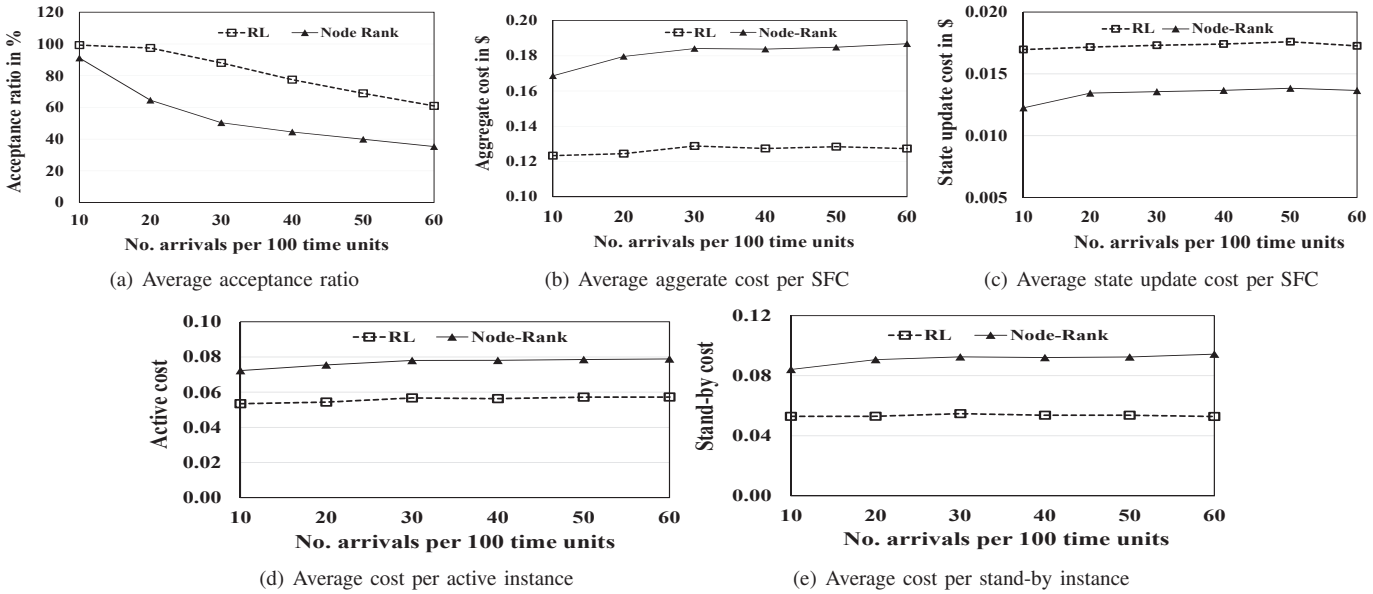
A. Simulation environment and settings

The number of DCs in the substrate network is varied from 40 to 60 with a connectivity probability fixed to 0.2. The CPU capacity of each DC, the cost of processing and transmitting 1GB of data at each DC and link respectively, the transmission delay on each link, and the processing delay of a packet at each NF follow uniform distributions as: U(4000 8000) Mhz, U(0.15, 0.22) $, U(0.05,0.12) $, U(2,5) ms, and U(0.045, 0.3) ms, respectively. Each request $r \in R$ is generated with a random source τ_s^r and destination τ_d^r, a transmission rate of U(400,4000) packets/sec and an end-to-end delay requirement of U(10,100) ms.

B. Results and discussion

The performance of the proposed RL based placement algorithm is compared with the state-of-the-art algorithm proposed in [15] and [17], and denoted as Node-Rank in this section, considering both online and offline scenarios. In the offline case, all service requests are known in advance and these, once admitted, do not leave the system. In the online case, the demands continuously arrive to the system with a given arrival distribution and with a defined life-time.

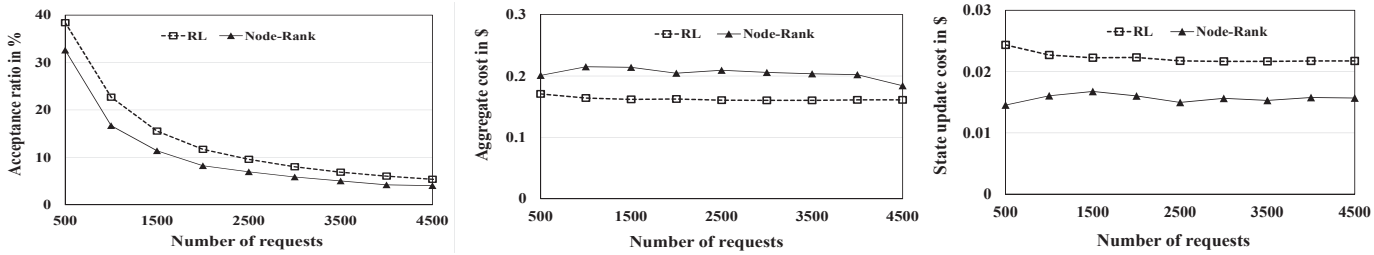
1) Online scenario: The results in Fig. 2 reveal the impact of the arrival rate on the algorithms' performance. From Fig. 2(a), RL results in a 27.7% improvement in terms of AR with an average value of 82.03% compared to 54.3% obtained by Node-Rank. This is because RL is able to jointly minimize the resource consumption due to: state update, active



(a) Average acceptance ratio (b) Average aggerate cost per SFC (c) Average state update cost per SFC

(d) Average cost per active instance (e) Average cost per stand-by instance

Fig. 2. Results of the online scenario with the arrival rates of requests varied from 10 to 60 requests per 100 time units for a total of 10000 time units



(a) Average acceptance ratio (b) Average aggerate cost per SFC (c) Average state update cost per SFC

Fig. 3. Results of the offline scenario in which the number of requests is varied from 500 to 4500 requests

instance placement and reserved backup resources. Moreover, RL is able to intelligently perform a trade-off between resource consumption and load balancing which accounts for the gain in performance as the arrival rate increases. From Fig. 2(b), RL and Node rank result in an average value of $0.13 and $0.18, respectively, in terms of aggregate placement cost, hence, RL shows a 30% improvement over Node-rank. This is because Node-rank greedily aims to minimize the state update cost without considering other costs such as the link resources of the active instance. This is evident from Fig. 2(c) where Node-rank results in a superior performance in terms of state update cost with an average value of $0.013 (approx. 22% improvement) compared to $0.017 for RL. And yet, RL results in up to a 27.3% and 41.2% improvement in terms of cost incurred for placement of active and stand-by instances, respectively, as shown in Fig. 2(d) and Fig. 2(e), further demonstrating the ability of RL to jointly optimize multiple attributes compared to the node ranking approach.

2) Offline: The results in Fig. 3 analyze the impact of the number of requests in the offline mode. A trend similar to the online case is observed, with Fig. 3(a) indicating that RL results in up to a 3% improvement in terms of AR, averaged across all requests size. From Fig. 3(b), RL results in an average placement cost of $0.16 per admitted request (20% improvement) compared to $0.20 for Node-Rank. From Fig. 3(c), Node-Rank results in close to a 30% improvement in terms of state update cost due to its greedy nature, yet performing poorly in the overall cost.

V. Conclusion

The paper has addressed the problem of stateful VNFs placement involving resource reservation for both active and stand-by VNF instances. A policy neural network with feasible state and action spaces for VNF placement with minimal cost has been proposed. From the simulation results, the proposed algorithm is able to keep a balance between load balancing and resource consumption, resulting in a high performance gain compared to a node ranking based algorithm, especially under congested scenarios. Simulation results reveal that the proposed algorithm results in up to a 27% and 30% improvement in terms of accepted requests and placement cost compared to the benchmark algorithm for some scenarios.

REFERENCES

[1] P. Kaliyammal Thiruvasagam, V. J. Kotagi, and C. S. R. Murthy, "The More the Merrier: Enhancing Reliability of 5G Communication Services With Guaranteed Delay," *IEEE Networking Letters*, vol. 1, no. 2, pp. 52–55, 2019.

[2] Y. Zhang, F. He, T. Sato, and E. Oki, "Network Service Scheduling with Resource Sharing and Preemption," *IEEE Transactions on Network and Service Management*, vol. 17, no. 2, pp. 764–778, 2020.

[3] G. Kibalya, J. Serrat, J. L. Gorricho, H. Yao, and P. Zhang, "A novel dynamic programming inspired algorithm for embedding of virtual networks in future networks," *Computer Networks*, vol. 179, no. May, p. 107349, 2020. [Online]. Available: https://doi.org/10.1016/j.comnet.2020.107349

[4] M. T. Beck, J. F. Botero, and K. Samelin, "Resilient allocation of service Function chains," *2016 IEEE Conference on Network Function Virtualization and Software Defined Networks, NFV-SDN 2016*, pp. 128–133, 2017.

[5] D. Cotroneo, L. De Simone, A. K. Iannillo, A. Lanzaro, R. Natella, J. Fan, and W. Ping, "Network function virtualization: Challenges and directions for reliability assurance," *Proceedings - IEEE 25th International Symposium on Software Reliability Engineering Workshops, ISSREW 2014*, pp. 37–42, 2014.

[6] W. Ding, H. Yu, and S. Luo, "Enhancing the reliability of services in NFV with the cost-efficient redundancy scheme," *IEEE International Conference on Communications*, vol. 1, pp. 1–6, 2017.

[7] S. R. Chowdhury, R. Ahmed, M. M. A. Khan, N. Shahriar, R. Boutaba, J. Mitra, and F. Zeng, "Dedicated Protection for Survivable Virtual Network Embedding," *IEEE Transactions on Network and Service Management*, vol. 13, no. 4, pp. 913–926, 2016.

[8] J. Fan, X. Gao, Z. Ye, K. Ren, C. Guan, and C. Qiao, "GREP: Guaranteeing reliability with enhanced protection in NFV," *HotMiddlebox 2015 - Proceedings of the 2015 ACM SIGCOMM Workshop on Hot Topics in Middleboxes and Network Function Virtualization, Part of SIGCOMM 2015*, pp. 13–18, 2015.

[9] X. Shang, Z. Li, and Y. Yang, "Rerouting Strategies for Highly Available Virtual Network Functions," *IEEE Transactions on Cloud Computing*, vol. PP, no. c, p. 1, 2019.

[10] S. Sharma, A. Kushwaha, A. Somani, and A. Gumaste, "Designing highly-available service provider networks with NFV components," *Proceedings - International Conference on Computer Communications and Networks, ICCCN*, vol. 2019-July, pp. 1–9, 2019.

[11] K. Karra and K. M. Sivalingam, "Providing Resiliency for Service Function Chaining in NFV systems using a SDN-based approach," *2018 24th National Conference on Communications, NCC 2018*, pp. 1–6, 2019.

[12] S. Bijwe, F. Machida, S. Ishida, and S. Koizumi, "End-to-end reliability assurance of service chain embedding for network function virtualization," *2017 IEEE Conference on Network Function Virtualization and Software Defined Networks, NFV-SDN 2017*, vol. 2017-Janua, pp. 1–4, 2017.

[13] L. Zhang, Y. Wang, X. Qiu, and H. Guo, "Redundancy mechanism of service function chain with node-ranking algorithm," *2019 IFIP/IEEE Symposium on Integrated Network and Service Management, IM 2019*, pp. 586–589, 2019.

[14] O. Soualah, M. Mechtri, C. Ghribi, and D. Zeghlache, "A link failure recovery algorithm for Virtual Network Function chaining," *Proceedings of the IM 2017 - 2017 IFIP/IEEE International Symposium on Integrated Network and Service Management*, pp. 213–221, 2017.

[15] B. Yang, Z. Xu, W. K. Chai, W. Liang, D. Tuncer, A. Galis, and G. Pavlou, "Algorithms for fault-tolerant placement of stateful virtualized network functions," *IEEE International Conference on Communications*, vol. 2018-May, 2018.

[16] W. Mao, L. Wang, J. Zhao, and Y. Xu, "Online Fault-tolerant VNF Chain Placement: A Deep Reinforcement Learning Approach," *IFIP Networking 2020 Conference and Workshops, Networking 2020*, pp. 163–171, 2020.

[17] G. Yuan, Z. Xu, B. Yang, W. Liang, W. Koong, D. Tuncer, A. Galis, G. Pavlou, and G. Wu, "Fault tolerant placement of stateful VNFs and dynamic fault recovery in cloud networks," *Computer Networks*, vol. 166, p. 106953, 2020. [Online]. Available: https://doi.org/10.1016/j.comnet.2019.106953