OptRec: An Efficient DRL-based SFC Reconfiguration Optimization Framework in Programmable Networks

Huaqing Tu[1], Ziqiang Hua[1], Qi Xu [1], Tao Zou [1], Huifeng Zhang[1], Hongli Xu[2], Zuqing Zhu[3]
[1]Zhejiang Lab, China
[2]School of Computer Science and Technology, University of Science and Technology of China, China
[3]School of Information Science and Technology, University of Science and Technology of China, China

***Abstract*—Service function chain (SFC) consists of multiple ordered network functions (*e.g.*, firewall, load balancer) and plays an important role in improving network security and ensuring network performance. Offloading SFCs onto programmable switches can bring significant performance improvement, but it suffers from unbearable reconfiguration delays, making it hard to cope with network workload dynamics in a timely manner. To bridge the gap, this paper presents OptRec, an efficient SFC reconfiguration optimization framework based on deep reinforcement learning (DRL). OptRec predicts future traffic and places SFCs on programmable switches in advance to ensure the timeliness of the SFC reconfiguration, which is a proactive approach. However, it is non-trivial to extract effective features from historical traffic information and ensure efficient and stable model training. To this end, OptRec introduces a multi-level feature extraction model for different types of features. Additionally, it combines reinforcement learning and autoregressive learning to enhance model efficiency and stability. Results of in-depth simulations based on real-world datasets show the average prediction error of OptRec is less than 3% and OptRec can increase the system throughput by up to 69.6%∼72.6% compared with other alternatives.**

***Index Terms*—Programmable Networks, Network Function, Service Function Chain, Reconfiguration, Deep Reinforcement Learning**

I. Introduction

As an essential component of today's network, network functions (NFs) [1], such as firewall and load balancer, are interconnected as a service function chain (SFC) [2] to ensure network security and improve network performance. In network function virtualization (NFV) [3], SFCs are implemented as software on general-purpose servers. However, the general-purpose servers are not dedicated forwarding devices, resulting in poor performance for software-based SFCs. For example, the maximum throughput of a general-purpose server is only O(10 Gbps) [4] and software-based SFCs incur latency ranging from 100 μs to 1 ms [5]. To improve SFC performance, recent research [5]–[7] offloads NFs onto programmable switches, which can bring significant SFC performance improvement in throughput (up to 6.4 Tbps [5]) and forwarding latency (less than 1 μs [4]).

This work was supported by the National Key Research and Development Project of China (No. 2022YFB2901503), the National Natural Science Foundation of China (No. U22A2005), Key Research Project of Zhejiang Lab (No. 2021LE0AC02). Corresponding authors: Ziqiang Hua, Tao Zou.

In real network scenarios, network workload changes drastically over time [2], resulting in overloaded NFs and poor quality of service. Thus, the NFs offloaded on programmable switches should be reconfigured so as to optimize network performance (*e.g.*, system throughput). For example, when network workload increases, it is necessary to offload more NFs on programmable switches to process more traffic; and when network load decreases, it is important to release redundant NFs to other applications (such as parameter aggregation [8]). Nevertheless, programmable switches suffer from poor flexibility [7], making it difficult to quickly reconfigure NFs when dealing with time-varying network workload.

Although there are already mature elastic scaling methods [9]–[11] to realize rapid NF reconfiguration in NFV scenarios, these methods cannot be applied to the programmable networks due to different underlying implementation mechanisms of NFs. To solve the reconfiguration problem of programmable switches, Xing *et al.* [7] proposes a runtime programmable framework, which enables partial reconfiguration of switch data planes at runtime without service disruption. Despite it supports seamlessly incorporating function changes at any time, the reconfiguration delay is unbearable. For example, when upgrading a telemetry application with $30K$ entries, the reconfiguration delay is about $10s$ [7]. Experimental results in various data centers show that more than 80% of traffic last for less than $10s$ [12]. Thus, such a long reconfiguration delay makes the SFC reconfiguration out-of-date. How to ensure the timeliness of NF configuration is still a challenge yet critical issue when dealing with network workload dynamics.

In this paper, we present OptRec, an SFC reconfiguration optimization framework based on deep reinforcement learning (DRL). To ensure the timeliness of NF configuration, OptRec aims to predict network workload changes in the future based on historical traffic information and reconfigure NFs on programmable switches in advance. However, achieving efficient SFC reconfiguration poses several challenges, even with access to historical traffic data. First, since separating prediction and reconfiguration procedures may introduce additional errors, it is crucial to achieve end-to-end NF reconfiguration. Second, due to the diversity of network information types (*e.g.*, traffic and switch), extracting effective features from network information is another challenge. Lastly, the training

process should be optimized to guarantee both efficiency and stability, enabling reliable predictions and reconfigurations.

To conquer these challenges, (1) OptRec adopts an end-to-end reconfiguration approach, utilizing multiple discrete action spaces to cover all the behaviors required for SFC deployment. This eliminates the need for additional manual rules and avoids introducing extra errors. (2) OptRec introduces a multi-level feature extraction model for different types of data and uses transformer encoder to obtain both a global overview and detailed features. (3) OptRec simultaneously utilizes reinforcement learning and autoregressive learning, which allows the model to learn from environmental feedback and proactively predict the traffic in the next time period, thereby improving model efficiency and stability.

We conduct extensive experiments based on real-world network topology and dataset. The results show that the average prediction error of OptRec is less than 3%. Moreover, OptRec can increase the system throughput by up to 69.6%∼72.6% compared with other alternatives.

II. Network Model and Problem Formulation

A. Network Model

We model a typical programmable network as a graph $G = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is the set of programmable switches and $\mathcal{E}$ is the set of links connecting programmable switches. Each programmable switch provides computational resources (*e.g.*, actions units) and memory resources (*e.g.*, TCAM and SRAM) for network function offloading. We use C_v to denote the resource capacity (including memory and computational resources) of programmable switch $v \in \mathcal{V}$. Moreover, let C_e be the bandwidth capacity of link $e \in \mathcal{E}$. As SFC requests will come and go as they run, we use $\mathcal{L}$, $\mathcal{O}$ and $\mathcal{B}$ to denote the set of existing SFC requests, the set of new arrival SFC requests and the set of expired SFC requests in a future time interval Δt. Let γ represent a SFC request in $\mathcal{L}$, $\mathcal{O}$ or $\mathcal{B}$. The SFC requirement and traffic size of each request γ are denoted as $\mathcal{F}_\gamma$ and $t(\gamma)$, respectively.

B. Problem Statement

Given a programmable network topology and history information of SFC requests, we aim to predict future traffic changes (that is, predict SFC request sets $\mathcal{L}$, $\mathcal{O}$ and $\mathcal{B}$ in the next time interval Δt) and pre-configurate network functions on programmable switches to cope with dynamic traffic changes. Since the network functions required by SFC requests in $\mathcal{L}$ have been configured and the SFC requests in $\mathcal{B}$ will expire, we only need to pre-configure network functions required by SFC requests in $\mathcal{O}$. The input includes a network topology $G = (\mathcal{V}, \mathcal{E})$, history SFC request information, the resource consumption (denoted as φ_f) of network function f when offloaded on programmable switches, link transmission delay (denoted as τ_e) on link $e \in \mathcal{E}$, network function processing delay (denoted as τ_v) on programmable switch $v \in \mathcal{V}$. Here, τ_e and τ_v can be obtained by periodically measured, and φ_f can be obtained through pre-testing. The output is two sets of binary decision variables, $\{x^v_{\gamma,f}\}$ and $\{y^p_\gamma\}$. Binary variable $x^v_{\gamma,f}$ indicates the mapping between network functions and programmable switches: $x^v_{\gamma,f} = 1$ if the network function $f \in \mathcal{F}_\gamma$ required by SFC request $\gamma \in \mathcal{O}$ is offloaded on programmable switch $v \in \mathcal{V}$; $x^v_{\gamma,f} = 0$ otherwise. Binary variable y^p_γ indicates whether the SFC request $\gamma \in \mathcal{O}$ will be routed through path $p \in \mathcal{P}_\gamma$, where $\mathcal{P}_\gamma$ is a candidate routing path set and can be constructed by the shortest k-path algorithm.

C. Constraints

The SFC reconfiguration problem should satisfy the following constraints:

1) *Routing Path Constraint:* Each SFC request $\gamma \in \mathcal{O}$ will be assigned a feasible path set, denoted as $\mathcal{P}_\gamma$, and be routed through at most one feasible path in $\mathcal{P}_\gamma$. This constraint can be formalized as follows:

$$\sum_{p \in \mathcal{P}_\gamma} y^p_\gamma \leq 1, \forall \gamma \in \mathcal{O} \tag{1}$$

2) *Network Function Offloading Constraint:* Once an network function $f \in \mathcal{F}_\gamma$ required by SFC request $\gamma \in \mathcal{O}$ is offloaded onto programmable switch $v \in \mathcal{V}$, the routing path $p \in \mathcal{P}_\gamma$ which passes through v should be selected. We use I^v_p to indicate whether routing path p passes through v (*i.e.*, I^v_p=1) or not (*i.e.*, $I^v_p = 0$). This constraint is expressed as:

$$x^v_{\gamma,f} \leq y^p_\gamma \cdot I^v_p, \forall \gamma \in \mathcal{O}, f \in \mathcal{F}_\gamma, v \in \mathcal{V}, p \in \mathcal{P}_\gamma \tag{2}$$

3) *Programmable Switch Resource Constraint:* Since the resource of each programmable switch is limited, the resource consumption cannot exceed its capacity. We use $I^v_{\gamma,f}$ to indicate whether the network function $f \in \mathcal{F}_\gamma$ required by SFC request $\gamma \in \mathcal{L} \cup \mathcal{B}$ was offloaded on programmable switch $v \in \mathcal{V}$ (*i.e.*, $I^v_{\gamma,f} = 1$) or not (*i.e.*, $I^v_{\gamma,f} = 0$) in the last time interval Δ_t. This constraint can be formulated as:

$$\begin{aligned} \sum_{\gamma \in \mathcal{L}} \sum_{f \in \mathcal{F}_\gamma} \varphi_f \cdot I^v_{\gamma,f} + \sum_{\gamma \in \mathcal{O}} \sum_{f \in \mathcal{F}_\gamma} \varphi_f \cdot x^v_{\gamma,f} \\ - \sum_{\gamma \in \mathcal{B}} \sum_{f \in \mathcal{F}_\gamma} \varphi_f \cdot I^v_{\gamma,f} \leq C_v, \forall v \in \mathcal{V} \end{aligned} \tag{3}$$

4) *Bandwidth Resource Constraint:* The total traffic on any link $e \in \mathcal{E}$ should not exceed its bandwidth capacity. We use I^e_γ to indicate whether the routing path of SFC request $\gamma \in \mathcal{L} \cup \mathcal{B}$ passed link $e \in \mathcal{E}$ (*i.e.*, $I^e_\gamma = 1$) or not (*i.e.*, $I^e_\gamma = 0$). We formulate this constraint as:

$$\begin{aligned} \sum_{\gamma \in \mathcal{L}} t(\gamma) \cdot I^e_\gamma + \sum_{\gamma \in \mathcal{O}} \sum_{e \in p : p \in \mathcal{P}_\gamma} t(\gamma) \cdot y^p_\gamma \\ - \sum_{\gamma \in \mathcal{B}} t(\gamma) \cdot I^e_\gamma \leq C_e, \forall e \in \mathcal{E} \end{aligned} \tag{4}$$

5) *End-to-End Transmission Constraint:* In order to ensure the quality of services in the network, each SFC request γ is associated with an end-to-end transmission threshold T_γ. The transmission delay of SFC request $\gamma \in \mathcal{O}$ consists of two parts: link transmission delay and network function processing delay on programmable

switches. The transmission delay of SFC request $\gamma \in \mathcal{O}$ should not exceed its transmission threshold T_γ. This constraint is expressed as:

$$\sum_{p\in\mathcal{P}_\gamma}\sum_{e\in p}\tau_e\cdot y_\gamma^p+\sum_{p\in\mathcal{P}_\gamma}\sum_{v\in p}\tau_v\cdot y_\gamma^p\le T_\gamma,\forall\gamma\in\mathcal{O} \quad (5)$$

D. Objective

The objective consists of two parts: the overall throughput and the number of reconfigurations. The overall throughput is the total traffic of SFC requests in $\mathcal{O}$ accepted by the network, which can be formulated as $\sum_{\gamma\in\mathcal{O}}\sum_{p\in\mathcal{P}_\gamma} y_\gamma^p\cdot t(\gamma)$. The number of reconfigurations is the total number of reconfigured NFs, which can be expressed as $\sum_{\gamma\in\mathcal{O}}\sum_{f\in\mathcal{F}_\gamma}\sum_{v\in\mathcal{V}} x_{\gamma,f}^v\cdot(1-I_v^f)$, where I_v^f denotes where there is a network function f belonging to an expired SFC request on programmable switch v ($I_v^f=1$) or not ($I_v^f=0$). When $I_v^f=1$, it means that there is no need to configure network function f on programmable switch v, since there is already a network function that is not used. The SFC reconfiguration problem is formalized as follows:

$$\max\ \eta_1\cdot\sum_{\gamma\in\mathcal{O}}\sum_{p\in\mathcal{P}_\gamma} y_\gamma^p\cdot t(\gamma)-\eta_2\cdot\sum_{\gamma\in\mathcal{O}}\sum_{f\in\mathcal{F}_\gamma}\sum_{v\in\mathcal{V}} x_{\gamma,f}^v\cdot(1-I_v^f)$$

$$s.t.\ Eqs.\ (1)\sim(5) \quad (6)$$

where η_1 and η_2 are positive weighted parameters, which can be set by network administrator to get a good trade-off among two parts. Since a larger throughput is desirable and a smaller number of reconfigurations is preferred, the two parts of the objective are connected using a minus sign.

III. Algorithm Design

If the SFC request sets $\mathcal{L}$, $\mathcal{O}$, and $\mathcal{B}$ are known, the network function reconfiguration problem can be solved by designing a approximate algorithm. However, due to the dynamic nature of SFC requests at future time intervals, traditional approximate algorithms are not suitable. Therefore, we propose OptRec, an efficient SFC reconfiguration optimization framework in programmable networks, which utilizes DRL to predict future traffic and proactively place network functions to ensure timely SFC configuration.

A. Overview

In this section, we introduce the workflow of OptRec. We define the interaction with the network to occur once or multiple times per *period*, with a fixed time interval Δt. During each period, the network state (*e.g.*, traffic, capacity usage of programmable switch) is collected at the granularity of each SFC. Next, this collected network state is preprocessed and fed into the transformer-based multi-level neural network of OptRec. The network structure consists of linear transformations, a pooling layer, and a backbone transformer encoder. This multi-level structure allows for the extraction of various features at a fine granularity, resulting in a more accurate representation of the network state. Employing the neural network, OptRec makes deployment decisions regarding whether to deploy the NFs of the SFC and where to place each NF. These decisions are made end-to-end without the need for additional artificial rules. The interaction process is illustrated in Fig. 1. During this process, training data and reward are collected and stored in the replay memory for model training. Once the training is completed, OptRec can provide efficient SFC deployment decisions during the running process.

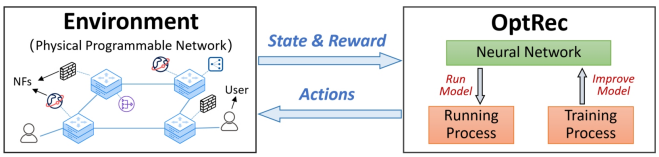


Fig. 1: Illustration of the Interaction between OptRec and Environment

B. Neural Network Architecture

*1) **Feature Aggregation Network**:* Since the collected data is heterogeneous, they need to be processed before sending into the model. For numerical features with a wide range of variations and uneven distribution, such as the queue length of SFC requests waiting for scheduling, we employ equal frequency binning to group values within specific ranges and perform one-hot encoding on them. For continuous values with limited amplitude, such as computation and memory capacity, we discretize them and apply one-hot encoding to enhance the significance of these features. In the context of NF information on a switch, there exist various states, including not deployed, deployed and used, and deployed but not used. To represent these categorical features, we introduce a set of trainable parameters known as NF state embeddings. Each state of a specific NF type is represented by a vector and learned during the end-to-end training process. Considering the possibility of multiple identical network functions on a switch, the state of each NF type is categorized into two situations: (1) all functions are in use; (2) there is at least one unused function.

Figs. 2 and 3 illustrate the networks used for extracting traffic features and switch features, respectively. The rectangles in the figures represent vectors. Apart from the one-hot encoded vector, the elements of the other vectors are real numbers. The inputs are initially fed into a preprocessing network to standardize the format for aggregation.

The aggregated traffic feature in Fig. 2 consists of three components: feature historical traffic, embedding of queued requests length, and embedding of live request number in network. Each component is a transformation of the initial input information. The historical traffic feature is obtained by extracting historical traffic information from a time series. The network utilizes a linear transformation with a Sigmoid-weighted linear unit (SiLU) activation function, which can be expressed as:

$$f_{\mathrm{FFN}_1}(\mathbf{x})=(\mathbf{x}\cdot\mathbf{W}+\mathbf{b})\otimes\sigma(\mathbf{x}\cdot\mathbf{W}+\mathbf{b}) \quad (7)$$

where $\otimes$ represents the element-wise product, and σ denotes the Sigmoid function. $\mathbf{W}$ is the model weight and $\mathbf{b}$ is the bias. The FFN_1 (feed-forward network) maps the original features to a fixed length of d_{hidden}. Similarly, the length of

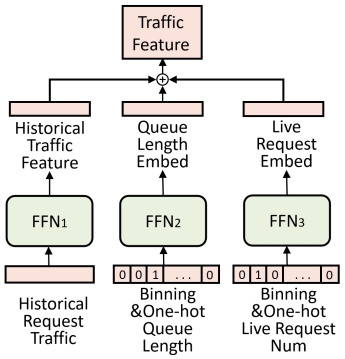


Fig. 2: Feature Engineering and Aggregation Network for Traffic Information

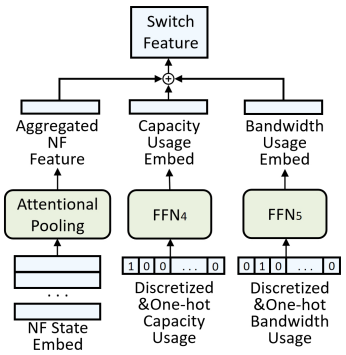


Fig. 3: Feature Engineering and Aggregation Network for Switch Information

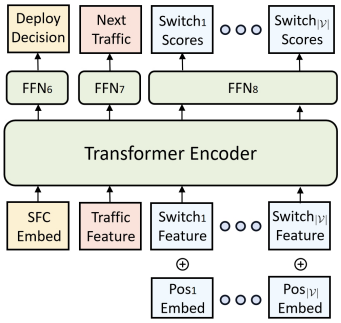


Fig. 4: Neural Network Architecture

queued SFC requests and the number of live SFC requests are binned, encoded, and transformed into vectors of length d_{hidden}. The FFN_2 and FFN_3 are both linear transformations. After aligning the features, they are summed up to form a vector that represents the aggregated traffic feature.

For switch feature extraction, we employ an attentional pooling network to capture NF information. The attentional pooling consists of two stages: the attention stage and the pooling stage. Let's define the shape of NF embeddings as $|\mathcal{F}| \times d_{\text{hidden}}$, where $\mathcal{F}$ represents the set of NF types. The attention stage can be represented as:

$$f_{\text{Atten}}(\mathbf{X}) = \text{ReLU}(\mathbf{X} \cdot \mathbf{W} + \mathbf{b}) \otimes \text{Softmax}_{\text{1st}}(\mathbf{X} \cdot \mathbf{V} + \mathbf{c}) \quad (8)$$

The subscript "1st" indicates the operation on the first dimension. ReLU and Softmax are both activation functions. The matrices $\mathbf{W}$ and $\mathbf{V}$ represent model weights of shape $d_{\text{hidden}} \times d_{\text{hidden}}$ and $d_{\text{hidden}} \times 1$, respectively. The vectors $\mathbf{b}$ and $\mathbf{c}$ represent biases. Next, the pooling stage sums the $|\mathcal{F}|$ reweighted vectors along each elemental dimension, which can be denoted as $f_{\text{Pool}}(\mathbf{X}) = \text{Sum}_{\text{1st}}(\mathbf{X})$. This results in an aggregated NF feature with a shape of $1 \times d_{\text{hidden}}$. The FFN_4 and FFN_5 are both linear transformations, and information of the capacity usage and bandwidth usage aggregated by switches are embedded and summed with the NF feature to produce the overall switch feature.

*2) **Backbone Network Structure**:* Once the traffic feature and switch feature are formed, along with an SFC embedding, they are input into a transformer encoder [13] to capture mutual correlation. The neural network architecture is depicted in Fig. 4.

The SFC embedding represents the type of SFC to be inferred for deployment and is a vector with trainable parameters of length d_{hidden}. The "$\text{Pos}_{|\mathcal{V}|}$ Embed" refers to the position embedding for $|\mathcal{V}|$-th switch. Since explicit topological features are not introduced by network architecture, we utilize the position embedding vector to learn the location features of switches in an end-to-end manner. The position embedding and the switch feature have the same shape and are fused together through element-wise addition.

The backbone network consists of a 2-layer transformer encoder. The input sequence is a matrix with dimensions $(|\mathcal{V}| + 2) \times d_{\text{hidden}}$. All feature tokens are crossed and propagated to capture the global network states, enabling communication and information exchange between the feature tokens. This process produces corresponding hidden states for the final outputs. $\text{FFN}_{6\text{-}8}$ are linear transformations. Notably, the network parameters of FFN_8 are shared by each switch feature token for parameter reuse.

There are three types of generated outputs. The deployment decision is a vector of length 2, where a higher value represents a different action: either *Idle* or *Deploy*. The term *Idle* signifies that there is no need to deploy new network functions for the SFC in the current state, indicating that the current deployment capacity is sufficient to handle the traffic requests for the next time period. The next traffic output is a scalar value that represents the predicted traffic bias for the next period based on the current traffic. Each switch score is a vector of length $|\mathcal{F}|$. The i-th element in the vector represents the placement weight of the i-th network function of the SFC on this switch. All switch scores form a matrix $\mathbf{Y}_{\text{score}} \in \mathbb{R}^{|\mathcal{V}| \times |\mathcal{F}|}$, and the placement locations of the network functions can be determined using $\text{Argmax}_{\text{1st}}(\mathbf{Y}_{\text{score}})$. If the number of network functions in an SFC is less than $|\mathcal{F}|$, any excess locations should be skipped.

C. Training and Running Process

The training process is to train the neural network of OptRec, and then running process is to produce SFC deployment decisions based on the trained model. To tackle the optimization problem with a multi-discrete action space, we utilize the dueling deep Q learning network (dueling DQN) [14] as the underlying framework for our algorithm. The action space encompasses decisions related to the behavior of deploying or not and the specific locations for each NF within the SFC. In addition, we incorporate multi-task learning to expedite training and improve interpretability. This approach involves autoregressive learning to predict traffic at the next period. Consequently, the overall loss function comprises the temporal difference (TD) loss associated with each action space and an error loss for traffic prediction.

According to the outputs, deploy decision and switch scores are two kind of action spaces to interact with networks, which are optimized by dueling DQN algorithm. The prediction of next period traffic bias is an autoregressive problem and is optimized by supervised learning.

Algorithm 1 Training Process

$\boldsymbol{\theta}, \boldsymbol{\theta}^- \leftarrow$ Initialize weights of online and target network;
for each *step* **do**
  $(\boldsymbol{S}, \boldsymbol{A}, r, \boldsymbol{S}') \leftarrow$ Choose a mini-batch from database;
  $loss \leftarrow 0$;
  for $a_i \in \boldsymbol{A} = \{a_{\text{decision}}, a_{\text{loc}_1}, ..., a_{\text{loc}_{|\mathcal{F}|}}\}$ **do**
    Accumulate $loss$ for action space a_i by weighted Eq. (9);
  Accumulate $loss$ for traffic prediction by weighted Eq. (10);
  Perform a gradient descent step on accumulated $loss$ and update network parameter $\boldsymbol{\theta}$;
  Periodically update target network parameters with $\boldsymbol{\theta}$;

Algorithm 2 Running Process

while *True* **do**
  for each SFC type $\omega_i \in \Omega$ **do**
    for $j = 1, 2, ..., N'$ **do**
      $\boldsymbol{S} \leftarrow$ Collect state from network;
      $a_{\text{decision}}, a_{\text{loc}_1}, a_{\text{loc}_2}, ..., a_{\text{loc}_{|\mathcal{F}|}} \leftarrow$ Get deploying decision and placing locations for ω_i from model $\boldsymbol{\theta}$ according to state $\boldsymbol{S}$;
      if a_{decision} = *Idle* **then**
        break;
      for $k = 1, 2, ..., |\mathcal{F}|^{\omega_i}$ **do**
        Place the k-th NF of ω_i according to a_{loc_k};
  Wait for the next deployment period;

The loss for an action a is defined as:

$$J_a(\boldsymbol{\theta}) = \mathbb{E}\Bigg[\Bigg(r + \alpha \cdot Q\big(\boldsymbol{S}', \underset{a}{\operatorname{argmax}}\ Q\big(\boldsymbol{S}', a|\boldsymbol{\theta}\big)|\boldsymbol{\theta}^-\big) - Q(\boldsymbol{S}, a|\boldsymbol{\theta})\Bigg)^2\Bigg] \tag{9}$$

where r represents the reward received after taking action a and is calculated as $\log(\eta_1 \cdot throughput + 1) - \log(\eta_2 \cdot queue\_length + 1) - \eta_3 \cdot place\_cost$. Here, η is a hyperparameter, and $place\_cost$ is a boolean value that indicates whether placement occurs. α is the discount factor. $Q(\boldsymbol{S}, a|\boldsymbol{\theta})$ is the current estimate of the action-value function. The parameters $\boldsymbol{\theta}$ and $\boldsymbol{\theta}^-$ represent the policy model weights and target model weights, respectively. $\boldsymbol{S}'$ represents the next state.

We utilize the mean square error (MSE) to optimize traffic prediction, as shown in Eq. (10). In this equation, $\hat{y}_\triangle$ represents the predicted traffic bias by the model, while y and y' represent the actual traffic in the current period and the next period, respectively.

$$J_y(\theta) = [\hat{y}_\triangle - (y' - y)]^2 \tag{10}$$

The entire training process is summarized in Algorithm 1. It is worth noting that the importance order of loss weights is deployment decision, deployment location and traffic prediction, which are set according to actual training status.

The running process, as depicted in Algorithm 2, is designed to continuously execute a series of steps for managing the deployment of SFCs. The algorithm first enters a perpetual loop. Within the loop, it iterates over each type of SFC, denoted as ω_i, from a SFC set denoted as Ω. For each type, the algorithm performs a number of placement attempts, with a maximum limit defined as N', based on the current state of the network. It is worth noting that $|\mathcal{F}|^{\omega_i}$ represents the number of NFs in a specific SFC type ω_i.

IV. Performance Evaluation

A. Simulation Settings

In the simulations, we select two typical and practical topologies as running examples. The first one is a data center topology, called Fat-Tree [15], which consists of 128 servers and 80 switches. This topology has been widely used by works like [16] to conduct experiment. The second one is a backbone network topology, called USANet [17], which consists of 24 switches and 47 links. For these two typologies, the data traces of abilene [18] are adopted to generate SFC requests. Note that there is no SFC information in the data traces. Thus, We generate the SFC information of each request according to the real SFCs [5]. Similar to [5], we set the resource capacity of switches: each programmable switch has 6400 action units and $50 \sim 100$ MB SRAM. An NF consumes $1000 \sim 1500$ action units and $6 \sim 12$ MB SRAM. Moreover, the throughput of a programmable switch is set to 6.4 Tbps. The link latency is set to $0.1\ \mu s \sim 0.5\ \mu s$ and the processing latency of programmable switch is set to $1\ \mu s$ [5].

B. Benchmarks

In the evaluation, we compare our proposed method with the following two DRL-based algorithms.

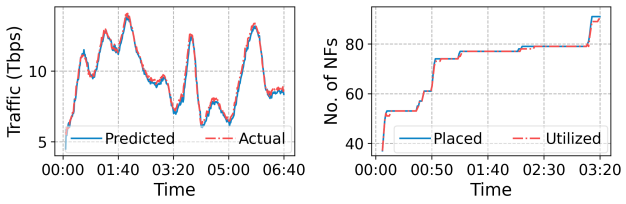
1) VNFPA [19] divides the network into regions and identifies the candidate region that requires optimization in each run. However, due to its threshold-based policy, VNFPA cannot achieve end-to-end network function deployment.

2) ADDPG [20] uses the remaining resources of links and nodes as input to predict the optimal placement locations for SFC requests. However, ADDPG does not consider time information and does not provide a decision on whether to deploy network functions in advance. This limitation hinders its ability to proactively deploy network functions.

Additionally, neither VNFPA nor ADDPG utilize NF-type information in their models. As a result, they are unable to perform targeted optimization for NF deployment, which can potentially lead to a decrease in placement accuracy.

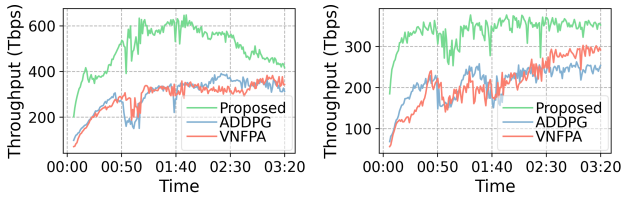
C. Simulation Results

(Exp #1) The Accuracy Performance of OptRec. We first evaluate the gap between the traffic predicted by OptRec and the actual traffic. Fig. 5(a) shows that the average error of traffic prediction is less than 3%, which reflects the high accuracy of OptRec's traffic prediction. Then, we evaluate the practical utilization of the pre-deployed NFs by OptRec. Fig. 5(b) shows that the average utilization rate of NFs deployed in advance is up to 99.62%.



(a) Traffic Prediction Accuracy (b) NF Placement Accuracy

Fig. 5: (Exp #1) Algorithm Performance



(a) Fat-Tree (b) USAnet

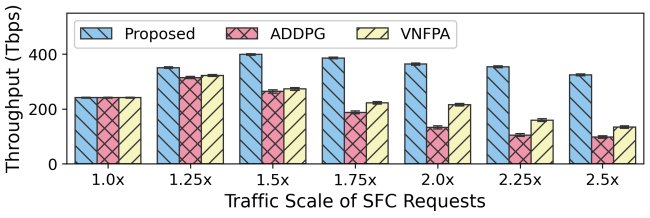
Fig. 6: (Exp #2) Throughput vs. Time

(Exp #2) Throughput Performance: We assess the throughput performance of the proposed method. Fig. 6 illustrates the throughput performance over time, while Fig. 7 demonstrates the decrease in throughput due to network congestion caused by increased traffic. The results indicate that OptRec achieves a significant improvement in average throughput compared to ADDPG and VNFPA, with an increase of 69.63% and 72.62%, respectively.

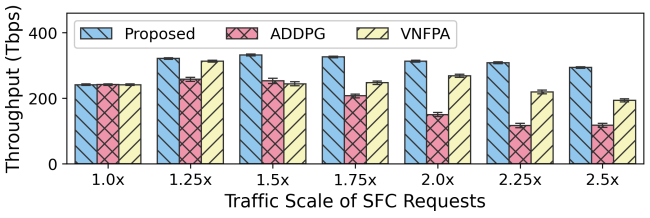
(Exp #3) Delay Performance: We assess the delay performance of the proposed method. Fig. 8 illustrates that OptRec achieves an average reduction in end-to-end delay of $0.17\ \mu s \sim 0.20\ \mu s$ compared to the benchmark methods. This result demonstrates the superior NF placement effectiveness of the proposed algorithm.

V. Conclusion

In this paper, we have presented OptRec, an efficient SFC reconfiguration optimization framework in programmable networks to deal with traffic dynamic. We propose a novel neural network algorithm based on deep reinforcement learning to reconfigurate NFs. Results of in-depth analyses based on real-world network topology and dataset show the efficacy of our algorithm compared with other solutions.

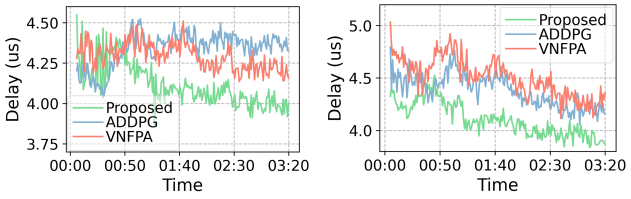


(a) Fat-Tree



(b) USAnet

Fig. 7: (Exp #2) Throughput vs. Traffic Scale of SFC Requests



(a) Fat-Tree (b) USAnet

Fig. 8: (Exp #3) Delay vs. Time

References

[1] H. Tu, G. Zhao, H. Xu, Y. Zhao, Y. Qiu, and L. Huang, "Rons: Robust network function services in clouds," *Computer Networks*, vol. 215, p. 109212, 2022.

[2] H. Tu, G. Zhao, H. Xu, Y. Zhao, and Y. Zhai, "A robustness-aware real-time sfc routing update scheme in multi-tenant clouds," *IEEE/ACM Transactions on Networking*, vol. 30, no. 3, pp. 1230–1244, 2022.

[3] K. Kaur, V. Mangat, and K. Kumar, "A review on virtualized infrastructure managers with management and orchestration features in nfv architecture," *Computer Networks*, vol. 217, p. 109281, 2022.

[4] D. Kim, Z. Liu, Y. Zhu, C. Kim, J. Lee, V. Sekar, and S. Seshan, "Tea: Enabling state-intensive network functions on programmable switches," in *Proc. SIGCOMM*, 2020, pp. 90–106.

[5] X. Chen, Q. Huang, P. Wang, Z. Meng, H. Liu, Y. Chen, D. Zhang, H. Zhou, B. Zhou, and C. Wu, "Lightnf: Simplifying network function offloading in programmable networks," in *Proc. IWQOS*. IEEE, 2021.

[6] Z. Liu, H. Namkung, G. Nikolaidis, J. Lee, C. Kim, X. Jin, V. Braverman, M. Yu, and V. Sekar, "Jaqen: A high-performance switch-native approach for detecting and mitigating volumetric ddos attacks with programmable switches," in *Proc. USENIX Security*, 2021.

[7] J. Xing, K.-F. Hsu, M. Kadosh, A. Lo, Y. Piasetzky, A. Krishnamurthy, and A. Chen, "Runtime programmable switches," in *Proc. NSDI*, 2022.

[8] C. Lao, Y. Le, K. Mahajan, Y. Chen, W. Wu, and A. Akella, "Atp: In-network aggregation for multi-tenant learning," in *Proc. NSDI*, 2021.

[9] G. P. Katsikas, T. Barbette, D. Kostić, J. G. Q. Maguire, and R. Steinert, "Metron: High-performance nfv service chaining even in the presence of blackboxes," *ACM Transactions on Computer Systems (TOCS)*, vol. 38, no. 1-2, pp. 1–45, 2021.

[10] J. Wang, G. Zhao, H. Xu, Y. Zhao, X. Yang, and H. Huang, "Trust: Real-time request updating with elastic resource provisioning in clouds," in *Proc. INFOCOM*. IEEE, 2022, pp. 620–629.

[11] W. Chen, Z. Wang, H. Zhang, X. Yin, X. Shi, and L. Sun, "Joint optimization of service function chain elastic scaling and routing," in *IEEE INFOCOM WKSHPS*. IEEE, 2020, pp. 1306–1307.

[12] X. Fan, H. Xu, H. Huang, and X. Yang, "Real-time update of joint sfc and routing in software defined networks," *IEEE/ACM Transactions on Networking*, vol. 29, no. 6, pp. 2664–2677, 2021.

[13] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[14] Z. Wang, T. Schaul, M. Hessel, H. Hasselt, M. Lanctot, and N. Freitas, "Dueling network architectures for deep reinforcement learning," in *International conference on machine learning*. PMLR, 2016.

[15] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," *ACM SIGCOMM computer communication review*, vol. 38, no. 4, pp. 63–74, 2008.

[16] S. Wang, H. Xu, L. Huang, X. Yang, and J. Liu, "Fast recovery for single link failure with segment routing in sdns," in *HPCC*. IEEE, 2019, pp. 2013–2018.

[17] N. Spring, R. Mahajan, and D. Wetherall, "Measuring isp topologies with rocketfuel," *ACM SIGCOMM Computer Communication Review*, vol. 32, no. 4, pp. 133–145, 2002.

[18] "abilene," http://sndlib.zib.de/home.action.

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

[20] N. He, S. Yang, F. Li, S. Trajanovski, L. Zhu, Y. Wang, and X. Fu, "Leveraging deep reinforcement learning with attention mechanism for virtual network function placement and routing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 34, no. 4, pp. 1186–1201, 2023.