IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 31 May 2024, accepted 2 July 2024, date of publication 8 July 2024, date of current version 18 July 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3424474

RESEARCH ARTICLE

InDS: Intelligent DRL Strategy for Effective Virtual Network Embedding of an Online Virtual Network Requests

T. G. KEERTHAN KUMAR[1,2], (Graduate Student Member, IEEE), SOURAV KANTI ADDYA[1], (Senior Member, IEEE), AND SHASHIDHAR G. KOOLAGUDI[1], (Member, IEEE)

[1]Cloud and Smart System Services Laboratory, Department of Computer Science and Engineering, National Institute of Technology Karnataka, Surathkal 575025, India
[2]Department of Information Science and Engineering, Siddaganga Institute of Technology, Tumakuru, Karnataka 572103, India

Corresponding author: T. G. Keerthan Kumar (keerthanswamy@gmail.com)

This work was supported by the National Institute of Technology Karnataka (NITK), Surathkal as per MOU.

ABSTRACT Network virtualization is a demanding feature in the evolution of future Internet architectures. It enables on-demand virtualized resource provision for heterogeneous Virtual Network Requests (VNRs) from diverse end users over the underlying substrate network. However, network virtualization provides various benefits such as service separation, improved Quality of Service, security, and more prominent resource usage. It also introduces significant research challenges. One of the major such issues is allocating substrate network resources to VNR components such as virtual machines and virtual links, also named as the virtual network embedding, and it is proven to be $\mathbb{NP}$-hard. To address the virtual network embedding problem, most of the existing works are *1*) Single-objective, *2*) They failed to address dynamic and time-varying network states *3*) They neglected network-specific features. All these limitations hinder the performance of existing approaches. This work introduces an embedding framework called **In**telligent **D**eep Reinforcement Learning (DRL) **S**trategy for effective virtual network embedding of an online VNRs (*InDS*). The proposed *InDS* uses an actor-critic model based on DRL architecture and Graph Convolutional Networks (GCNs). The GCN effectively captures dependencies between the VNRs and substrate network environment nodes by extracting both network and system-specific features. In DRL, the asynchronous advantage actor-critic agents can learn policies from these features during the training to decide which virtual machines to embed on which servers over time. The actor-critic helps in efficiently learning optimal policies in complex environments. The suggested reward function considers multiple objectives and guides the learning process effectively. Evaluation of simulation results shows the effectiveness of *InDS* in achieving optimal resource allocation and addressing diverse objectives, including minimizing congestion, maximizing acceptance, and revenue-to-cost ratios. The performance of *InDS* exhibits superiority in achieving 28% of the acceptance ratio and 45% of the revenue-to-cost ratio by effectively managing the network congestion compared to other existing baseline works.

INDEX TERMS Network virtualization, deep reinforcement learning, resource utilization, network features, congestion, acceptance ratio, virtual network embedding.

I. INTRODUCTION

NNetwork Virtualization (NV) is a pivotal solution that significantly contributes to effectively managing the next-generation Internet architecture. It decouples the

The associate editor coordinating the review of this manuscript and approving it for publication was Nitin Gupta.

physical network resources, enhancing the future Internet's flexibility, isolation, agility, and scalability [1], [2], [3]. Due to these benefits, Service Providers (SPs) try to separate physical resources logically into various distinct Virtual Network Requests (VNRs) to cater to heterogeneous demands. This allocation problem is referred to as Virtual Network Embedding (VNE), and is known to be $\mathbb{NP}$-hard [4]. Figure 1 depicts a sample VNR with three Virtual Machines (VMs) and three Virtual Links (VLs). The numeral associated with VMs and VLs signifies their computational and bandwidth resource demands. The VM $v_{1,3}$ is associated with a computational resource demand of 8 units and is pertained as a Computational Resource Block (CRB) of that VM. A unit of CRB resource infers 1 core CPU and 512 MB RAM. Many researchers adopt such sizing in works [5], [6]. Similarly, the VL between the VMs $v_{1,2}$ and $v_{1,3}$ has a minimum bandwidth requirement of 11 *Gbps* for successful communication. In VNE, the acceptance ratio analyzes the success rate of accepting VNRs, the revenue-to-cost ratio examines the profitability of virtualized network services, and the congestion ratio checks network resource consumption and efficiency. These primary objectives are critical for optimizing network performance, maximizing revenue-to-cost ratios, and increasing long-term profitability for SPs [2], [5], [7], [8].

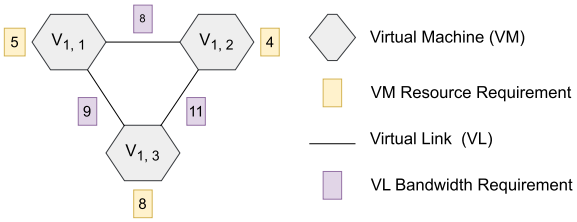

FIGURE 1. An instance of a VNR containing three VMs and three VLs, along with their corresponding CRB and bandwidth demands.

In this context, various traditional approaches, such as heuristic, meta-heuristic, and exact solution techniques, have been proposed to address the VNE problem. In [9], authors Zhang et al. introduced a comprehensive ranking method for effective load distribution and maximizing the acceptance ratio. In work [2], authors developed a matching game-based VNE strategy to lower the prevailing cost of the embedding. However, these works fail to capture accurate network dependencies due to a lack of topological attributes, resulting in degraded performance [2], [9]. Further, in [10], the authors introduced a Google PageRank-based ranking mechanism with limited network attributes. However, this approach is limited to smaller scenarios. Further, in [11], the authors developed a meta-heuristic-based VNE strategy for maximizing the acceptance ratio. Randomness in the solution leads to inferior performance. Most of the above approaches only focus on maximizing the acceptance and revenue-to-cost ratios metrics and neglect network congestion avoidance, which leads to poor substrate resource utilization and decreased network performance. To tackle this, a few researchers in [12] and [13] developed congestion-aware heuristic strategies. However, these approaches lack dynamic network adaptability and are less scalable. On the other hand, few learning-based approaches exhibit a lack of network performance due to the negligence of the network features during the training [14], [15]. Moreover, all these solution approaches exhibit the following impediments. (*i*.) Most heuristic strategies fail to guarantee a global optimal solution. (*ii*.) Few exact strategies face scalability issues. (*iii*.) Existing meta-heuristic strategies are complex and time-consuming (*iv*.) They lack in handling dynamic workloads during their lifecycle. (*v*.) Most learning-based solutions neglect network-specific parameters. To overcome these limitations, in this work, we propose a framework called intelligent DRL Strategy for effective VNE over online VNRs (*InDS*). It makes use of hybrid *Deep Reinforcement Learning* (DRL) and *Graph Convolutional Networks* (GCNs) based strategy for effective one-stage VNE. *InDS* tries to enhance the acceptance ratio revenue-to-cost ratio and avoid network congestion, considering both network and system resources featured during training. The RL provides excellent decision-potentiality capabilities by efficiently cooperating between the learning agent and the input environment, thereby maximizing the reward for the agent [15]. The GCN is a specific type of graph neural network that uses convolutional operations on graph-structured data to dynamically extract network and system-specific features of the SN and VNRs [16], [17]. Therefore, we use the benefits of both RL and GCN to aggregate network features, such as degree and betweenness centrality, along with system characteristics, such as CRB and bandwidth resources. These network features help understand the network dependencies effectively to achieve the desired multiple objectives. The significant contribution of this work is highlighted below.

- **Hybrid DRL and GCN based Dynamic Framework**: This work introduces an intelligent embedding framework named *InDS*. It uses DRL with an Asynchronous Advantage Actor-Critic (A3C) architecture to make sequential decisions considering system and network features in a dynamic environment. Incorporating GCN helps capture network features of online VNRs and the underlying SN. *InDS* evaluates four main attributes such as CRB, bandwidth, degree, and betweenness centrality. The DRL agent is trained on a multiple of this four-dimensional network environment. We use synthetic data that closely resembles real-world data to train the model. The produced data allows for successful training and regulated testing settings. Thereby *InDS* try to maximize the acceptance and revenue-to-cost ratios by effectively handling network congestion.
- **Adaptive Resource Allocation Policies**: We construct learning policies during parallel policy gradient training using input states from the dynamically varying network captured using GCNs. We also define a fair multi-objective reward function that helps to make informed decisions about the VM allocation over the substrate server based on the probability reward.

Followed by Breadth-First Search (BFS) [18], shortest path link embedding is subjected to various constraints.

- **Performance Evaluation and Baselines**: We use *alib utility tool* to implement *InDS*, and the source code of our implementation is made available at [19]. We run exhaustive simulations to evaluate *InDS* performance and compare with four different baseline strategies such as (*i*.) Node Ranking Measurement (*VNE-NRM*) [11] (*ii*.) Automatic reinforcement based VNE (*A3C-GCN*) [14] (*iii*.) Modified Worst Fit Strategy (*VNE-MWF*) [6] (*iv*.) Distributed Parallel Genetic Algorithm (*DPGA*) [20]. The results of the tests demonstrate that *InDS* performs better than other baselines by scoring an improvement of 28% in acceptance and 45% in revenue-to-cost ratio.

The remainder of the content is classified as follows. Section II delivers an exhaustive literature review on the various traditional and learning-based techniques to address the VNE problem. Section III provides the proposed system model and different associated modules and their functionalities. The formulation of *InDS* objectives and the associated constraints are discussed in Section IV. Section V details the proposed DRL and GCN-based solution approach and training strategy adopted in *InDS*. Section VI details the testing and performance analysis of the *InDS* with distinct baseline approaches. Finally, the conclusion and future scope of the *InDS* are discussed in Section VII.

II. RELATED WORKS

This section explores the literature review of several existing VNE solution strategies. We primarily classify the embedding strategies into traditional and learning-based techniques. The VNE problem is divided into two intractable problems: (*i*.) VM-embedding and (*ii*.) VL-embedding. These two are solved either in a coordinated or uncoordinated technique. Compared to uncoordinated approaches, a coordinated embedding technique between the VM and VL embedding problems can improve overall performance, mainly when nearby nodes are close together [2], [5]. Based on VM and VL embedding coordination, these techniques are either one-stage or two-stage. The former does both embedding simultaneously, whereas the latter performs them sequentially. This section also elaborates on the type of VNR request and the features the various solution strategies consider.

A. TRADITIONAL STRATEGIES

The various researchers developed traditional embedding models based on heuristics, meta-heuristics, and exact solution approaches. In this context, the authors in [21] and [22] proposed exact solutions emphasizing enhancing the acceptance ratio. The former developed a coordinated two-stage approach by primal-dual investigation of the Mixed Integer Linear Programming (MILP) to get a near-optimal solution over static VNRs. The latter suggested a dynamic one-stage VHub method to model VNE as a p-hub median problem to enhance the revenue-to-cost ratio. These exact approaches are limited to smaller test scenarios and are computationally complex. To address these limitations, in [11], authors Zhang et al. presented a coordinated two-stage heuristic approach with diverse resource restrictions to enhance the acceptance and revenue-to-cost ratios over dynamic requests. However, the lack of a sophisticated embedding mechanism results in inferior performance. Further, in [2], the authors developed a one-to-many matching game-based two-stage VNE strategy for cost minimization over online VNRs. However, due to the preference generation mechanism, it takes longer computational time for more prominent test cases. Moreover, all the above approaches neglect the network features and congestion avoidance, resulting in degraded performance. To overcome the above shortcoming, authors Tg et al. in [5] developed a two-stage heuristic-based multi-attribute embedding approach over static VNRs considering both network and system features to maximize the revenue-to-cost ratio. However, a greedy embedding mechanism results in a local-optimal solution. Further, in work [20], the authors developed a meta-heuristic-based Genetic Algorithm (GA) strategy to maximize the acceptance and revenue-to-cost ratios over static VNRs. However, randomness in the solution approach leads to poor resource utilization, and it has more computational overhead. Later, even though these approaches considered few network attributes, they neglected congestion avoidance over a network, resulting in degraded performance. In [13], the authors developed a heuristic-based perturbation approach to minimize the congestion over a link and enhance the overall revenue-to-cost ratios. Due to homogeneous link bandwidth over VNRs, its wider adoption was limited. Further, in work [24], the authors introduced a one-stage heuristic strategy to minimize the cost, energy, and congestion during embedding over dynamic demands subjected to only system-specific constraints. In [12], the authors, Pham et al. designed a two-stage heuristic technique. It entails greedy node mapping to accomplish the goals of congestion control and resource optimization. However, the lack of sophisticated network parameters results in lower performance. Most of the above approaches often exhibit the following impediments. (*i*.) These strategies are static and rule-based (*ii*.) Lack of adaptability for dynamic environments, (*iii*.) Neglected network parameters and (*iv*.) Existing approaches are time-consuming, which results in overall inferior performance.

B. LEARNING-BASED STRATEGIES

To overcome the limitations of the traditional approaches, a few authors proposed learning-based embedding strategies to accelerate the revenue-to-cost ratio, acceptance ratio, and load balancing by effectively utilizing the SN resources. In this context, in [23], the authors Yao et al. proposed an RL-based policy network strategy for decision-making during VM embedding. It uses a policy gradient to automatically achieve optimization by using past data from dynamic VNRs. However, lacking network features during training reduces network performance and is limited to smaller scenarios. The

TABLE 1. Overview of the literature review on related VNE studies.

Research Work	Request Type		Embedding Strategy Type		Co-ordination Stages		Features Considered	
	Static	Dynamic	Traditional	Learning	One	Two	System	Network
Hu *et al.* [21] (2014)	✓	✗	✓	✗	✗	✓	✓	✗
Shanbhag *et al.* [22] (2015)	✗	✓	✓	✗	✓	✗	✓	✗
Yan *et al.* [13] (2016)	✓	✗	✓	✗	✗	✓	✓	✗
Zhang *et al.* [11] (2017)	✗	✓	✓	✗	✗	✓	✓	✗
Yao *et al.* [23] (2018)	✗	✓	✗	✓	✗	✓	✓	✗
Pham *et al.* [24] (2019)	✗	✓	✓	✗	✓	✗	✓	✗
Dolati *et al.* [25] (2019)	✗	✓	✗	✓	✓	✗	✓	✗
Yan *et al.* [14] (2020)	✗	✓	✗	✓	✗	✓	✓	✗
Pham *et al.* [12] (2020)	✗	✓	✓	✗	✗	✓	✓	✗
Nguyen *et al.* [20] (2020)	✓	✗	✓	✗	✗	✓	✓	✓
Zhang *et al.* [15] (2022)	✗	✓	✗	✓	✗	✓	✓	✓
Satpathy *et al.* [2] (2022)	✗	✓	✓	✗	✗	✓	✓	✗
Tg *et al.* [5] (2023)	✓	✗	✓	✗	✗	✓	✓	✓
Duan *et al.* [26] (2024)	✗	✓	✗	✓	✗	✓	✓	✗
InDS (**Proposed**)	✗	✓	✗	✓	✓	✗	✓	✓

work in [25] introduces an embedding approach based on DRL to maximize the revenue-to-cost ratio. It uses Deep Neural Networks (DNNs) for feature extraction. However, DNN failed to process the graph structure directly, making the model complex and limiting its wider adoption. To overcome the limitations of DNN, in [14], the authors proposed a strategy combining GNN with DRL to handle automatic VNE problems. It uses parallel training using the RL technique to enhance the acceptance and revenue-to-cost ratios. However, it is limited to smaller scenarios, and negligence of the network features results in degraded performance. In work [15], the authors developed a GCN and RL-based model with a self-defined fitness matrix and a fitness value to minimize substrate resource fragmentation and improve the revenue-to-cost ratio by considering the system and limited network features. Moreover, the learning strategy adopted heavily depends on the system features, which decays its performance. On the other hand, in [27], the authors developed a DRL-based embedding strategy considering limited network features to maximize the revenue-to-cost ratio for small test case scenarios. Later, in [26], the author Duan and Wang introduced a two-stage VNE approach using the proximal policy optimization for embedding policy generation. Additionally, it uses a composite feature extraction technique that blends features extracted with GCN to enhance the acceptance and revenue-to-cost ratios. However, these learning-based approaches exhibit the following pitfalls.(*i*.) Neglected network features during the training, (*ii*.) Limited for smaller cases, and (*iii*.) Failed congestion avoidance over a network (*iv*.) Most of the reward functions are specific to a single objective. These things result in poor SN utilization and degraded performance.

C. TAKEAWAYS OF THE LITERATURE REVIEW

The following are some key takeaways from the reviewed literature.

- **Inadequate adaptability to dynamic environments**. Most existing approaches are static and based on predefined rules, resulting in inefficiency in handling dynamic heterogeneous VNRs.
- **Limited scalability**. Most traditional heuristic techniques stumble to handle large-scale scenarios efficiently due to dependency on iterative processes-based solution strategies.
- **Computational complex**. Most meta-heuristic and exact approaches are computationally expensive for more extensive scenarios, making them impracticable for real-time deployments. Moreover, most meta-heuristic techniques result in sub-optimal solutions.

- **High computational cost**. Most of the exact formulations are computationally expensive, requiring significant computational resources and time. It results in delayed decision-making and inferior performance.
- **Negligence of Network Features**. The existing learning-based approaches inadequately explored critical network features during training. It results in models lacking a thorough understanding of network dynamics, leading to degraded network performance.
- **Limited Focus of Reward Functions**. Most reward functions are designed to achieve a specific single goal. This restricted focus limits the solution techniques to optimize for larger network performance measures, perhaps leading to sub-optimal outcomes.

To address the potential risks in the existing literature, we propose an RL-based framework called InDs for effective VNE for online VNRs. It uses a hybrid DRL and GCNs-based strategy that considers diverse network and system features during training for effective VNE. Finally, Table 1 summarizes the examined literature based on various parameters such as request type (Static/Dynamic), embedding strategy type (Traditional/Learning), co-ordination stages (One/Two), and features considered (System/Network).

III. ARCHITECTURE OF InDS

The proposed model is based on a Hybrid DRL and GCN-based dynamic framework called *InDS*, which has substantial advantages over existing approaches. To handle complicated network scenarios dynamically and efficiently, it combines DRL-based A3C architecture and GCN. The A3C design improves real-time decision-making. GCN captures intricate network properties of online VNRs and the underlying SN, focusing on essential attributes, including CRB, bandwidth, degree, and betweenness centrality. Training using synthetic data that simulates real-world circumstances ensures successful training and testing environments. This hybrid approach offers a robust, scalable, and intelligent solution for VNE, outperforming traditional block-based and modular structures in handling dynamic network environments. The proposed *InDS* architecture is depicted in Figure 2, and the following sections outline the functionalities of each component included in the *InDS*.

A. INPUT ENVIRONMENT

It includes information about the state of the substrate network, such as available resources, used resources, and heterogeneous demands for virtual network requests. The environment simulates the conditions in which the agent operates and responds to the agent's actions by updating the network state and providing feedback as rewards.

1) VIRTUAL NETWORK REQUESTS (VNRs)

The total VNRs arrived for processing are captured in $\mathbb{G}_v = \{\mathbb{G}_1, \mathbb{G}_2, \ldots, \mathbb{G}_i, \ldots\}$, with each VNR $\mathbb{G}_i \in \mathbb{G}_v$ is represented as $\mathbb{G}_i = (\mathbb{N}_i, \mathbb{L}_i)$. The variable $\mathbb{N}_i$ represents a set of VMs, and $|\mathbb{N}_i|$ provides a total VM count in a VNR

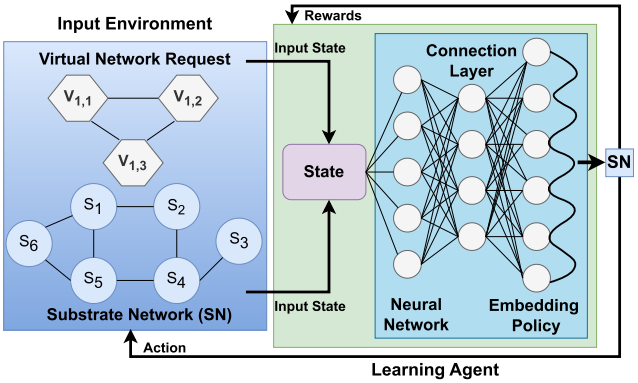

FIGURE 2. The proposed architecture of *InDS* for VNE problem.

$\mathbb{G}_i$. The resource demand of a VM $v_{i,j} \in \mathbb{N}_i$ is captured in variable $d(v_{i,j})$, and it is quantified in CRBs. A unit CRB corresponds to one CPU core and 512 MB of RAM, and such sizing is adopted by many of the SPs [2], [28]. For instance, Figure 1 depicts a sample VNR in which a VM $v_{1,2}$ is associated with the numeral four, indicating a requirement for 4 CPU cores and 2 GB of RAM. In addition, the set $\mathcal{V} = \bigcup_{i=1}^{|\mathbb{G}_v|} \mathbb{N}_i$ captures the total number of VMs to be embedded across the VNRs. Additionally, a variable $\mathbb{L}_i$ captures a set of VLs, where $|\mathbb{L}_i|$ gives the total VL count. The variable $d(e^i_{j,j'})$ captures the bandwidth demand of a VL $e^i_{j,j'}$ between the VMs $v_{i,j}$ and $v_{i,j'}$. Since *InDS* dealing with online VNRs, the fluctuating resource demand of VMs and VLs over a time slot t is captured in $\overline{d}_t(v_{i,j})$ and $\overline{d}_t(e^i_{j,j'})$ respectively.

2) SUBSTRATE NETWORK (SN)

The SN formally defined as $\mathbb{G}_s = (\mathbb{N}_s, \mathbb{L}_s)$. In SN, $\mathbb{N}_s$ and $\mathbb{L}_s$ represent the substrate server and substrate link set, respectively. Each server $s_k \in \mathbb{N}_s$ has initial available resource capacity captured in $a(s_k)$. For example, Figure 3 shows an instance of the SN comprising six servers and seven physical links. The servers such as s_1 and s_2 possess computational resources quantified as 32 and 22 units of CRBs, respectively. Besides, each SN link $e_l \in \mathbb{L}_s$ is associated with an available bandwidth capacity represented as $a(e_l)$. For illustration, in Figure 3, the numeral 23 over the link connecting servers s_1 and s_2 denotes its available total bandwidth capacity. It is important to note that the numerical values in Figure 3 are provided for illustrative purposes only. One of the fundamental aspects of VNE involves the successful embedding of VLs. To achieve this, we focus on finding appropriate substrate paths that meet the bandwidth demands of VLs. We denote the set of all possible explicit paths between two servers s_k and $s_{k'}$, hosting VMs $v_{i,j}$ and $v_{i,j'}$, respectively, as $Ps_k, s_{k'}$. A typical path $p_{s_k,s_{k'}} \in Ps_k, s_{k'}$ within this set can effectively embed VL $e^i_{j,j'}$, *iff* the available bandwidth $a(e_l)$ on each link $e_l \in p_{s_k,s_{k'}}$ exceeds or equals the bandwidth demand $\overline{d}_t(e^i_{j,j'})$ of the VL $e^i_{j,j'}$. For reference, we have summarized the key notations used and their specifications in Table 2.

TABLE 2. Notations and its specification.

Notation	Specification
$\mathbb{G}_s = \{\mathbb{N}_s, \mathbb{L}_s\}$	Substrate network
$\mathbb{G}_v = \{\mathbb{G}_1, \ldots, \mathbb{G}_i, \ldots\}$	Set of VNRs
$\mathbb{N}_s = \{s_1, s_2, s_k, \ldots\}$	Set of substrate servers
$\mathbb{L}_s = \{e_1, e_2, e_l, \ldots\}$	Set of substrate links
$\mathbb{P}_{s_k, s_{k'}}$	Set of all substrate paths
$p_{s_k, s_{k'}}$	A particular path in $\mathbb{P}$
$\mathbb{G}_i = \{\mathbb{N}_i, \mathbb{L}_i\}$	i^{th} VNR
$\mathbb{N}_i = \{v_{i,1}, v_{i,2}, \ldots\}$	Set of VMs of i^{th} VNR
$\mathbb{L}_i = \{e^i_{1,1}, \ldots, e^i_{j,j'}, \ldots\}$	Set of VLs i^{th} VNR
$\mathbb{V}$	Total VMs to be embedded from $\mathbb{G}_v$
k	Indexing used for servers
i	Indexing used for VNRs
j	Indexing used for VMs
$v_{i,j}$	j^{th} VM of VNR $\mathbb{G}_i$
$\mathbb{C}(\mathbb{G}_i)$	Total cost in dollar for embedding $\mathbb{G}_i$
$\mathbb{R}(\mathbb{G}_i)$	Total revenue earned in dollar from $\mathbb{G}_i$
Γ_i	Revenue-to-Cost Ratio
s_k	k^{th} server of SN $\mathbb{G}_s$
$\overline{d}_t(v_{i,j})$	The CRB demand of a VM $v_{i,j}$ at time t
$e^i_{j,j'}$	Denote a VL connects VMs $v_{i,j}$ and $v_{i,j'}$
$f(s_k, s_{k'})$	Flow of traffic from server s_k to $s_{k'}$
R	Upper bound in maximum link utilization
$\overline{d}_t(e^i_{j,j'})$	The bandwidth demand of a link $e^i_{j,j'}$ at time t
$a^{i,j}_k$	Action of assigning VM $v_{i,j}$ to a server s_k
s^p_t	Set of state features of SN
s^v_t	Set of state features of VNR
$a(e_l)$	Available bandwidth resource at link e_l
$a(s_k)$	Available CRB resource at server s_k
$\mathtt{D}(n)$	Provides degree centrality of a node 'n'
$\mathcal{B}_s(v_{i,j})$	Provides bandwidth strength of a VM $v_{i,j}$
$\mathcal{B}_s(s_k)$	Provides bandwidth strength of a server s_k
$\mathtt{B}(n)$	Provides betweenness centrality of a node 'n'
$\mathbb{C}_r$	Congestion Ratio
$\chi(v_{i,j}, s_k)$	Node indicator variable
γ_t	Discount factor between 0-1
$\chi(e^i_{j,j'}, p_{s_k, s_{k'}})$	Link indicator variable
β	Decaying attribute set to 0.5
l^a_r	The actor learning rate set to 0.00025
l^c_r	The critic learning rate set to 0.0025
d_e	Decay factor

B. REINFORCEMENT LEARNING (RL) ENTITIES

An RL framework comprises four major entities, such as (*i.*) State, (*ii.*) Action, (*iii.*) Reward and (*iv.*) Learning agent. The descriptions of each entity are provided below.

1) STATE DEPICTION

The state $s_t = \{s^p_t, s^v_t\}$ represents input information for an agent from the input environment, and it operates as the raw input for the subsequent stage. The s^p_t and s^v_t are the real-time representations of diverse SN and VNR features in vector form, respectively. The SN feature includes *maximum CRB capacity*, *maximum bandwidth strength of a server*, *available CRB capacity*, *available bandwidth strength of a server*, *betweenness centrality*, and *degree centrality*. Similarly, the features of the VNRs include *CRB demand of a VM*, *total bandwidth demand*, *degree centrality of a VM*, and *betweenness centrality of a VM*. These features are the most significant and widely regarded for a VNE [5], [6], [10] and a description of each feature is as follows.

- **Degree centrality**: The degree centrality $\mathtt{D}(n)$ of a specific node 'n' represents the number of nodes neighboring to it [29], and it is represented as $\mathtt{D}(n) = |adj(n)|$. The function $adj(.)$ gives the set of neighboring nodes. Nodes with a high degree of centrality are frequently considered more prominent in the network since they have more direct connections for interactions with other nodes.
- **Betweenness centrality**: The betweenness centrality $\mathtt{B}(n)$ of a node 'n' of a network is the proportion of shortest paths passing through that node [29] and the same is computed as given by:

$$\mathtt{B}(n) = \sum_{n \neq n' \neq n''} \frac{\sigma_{n'n''}(n)}{\sigma_{n',n''}} \tag{1}$$

  A node with a high betweenness centrality is located on many of the network's shortest pathways and serves as a crucial intermediary. It also supports efficient communication and the flow of information throughout the network.
- **Maximum server capacity**: The maximum server capacity of a server s_k refers to the maximum available CRB resources $a(s_k)$ to cater to the needs of the VMs over a SN [5].
- **Bandwidth strength of a server**: The bandwidth strength $\mathcal{B}_s(s_k)$ of a server s_k in a network often refers to the server s_k's ability to manage network traffic in terms of bandwidth requirement by the VLs [5] and which is defined as:

$$\mathcal{B}_s(s_k) = \sum_{\forall s_{k'} \in adj(s_k)} a(e_l) \tag{2}$$

- **Residual server capacity**: The residual server capacity of a server s_k refers to the amount of computational resources left on a server s_k after allocating CRB resources to fulfill the demands of the VMs that are embedded over it [5].
- **Residual bandwidth strength**: A server's $s_k \in \mathbb{G}_s$ residual bandwidth strength refers to the remaining bandwidth capacity available for data transmission across the substrate server s_k after embedding certain VNRs on it [5].
- **CRB demand**: The VM $v_{i,j}$ CRB demand $\overline{d}_t(v_{i,j})$ denotes the computational resources required by a VM $v_{i,j}$ of a VNR $\mathbb{G}_i$ [5].
- **Total Bandwidth demand**: The bandwidth demand of a VM $\mathcal{B}_s(v_{i,j})$ represents the total bandwidth required by a VM $v_{i,j} \in \mathbb{G}_i$ for establishing successful communication with its neighbor [5], and the same can be stated as below:

$$\mathcal{B}_s(v_{i,j}) = \sum_{\forall v_{i,j'} \in adj(v_{i,j})} \overline{d}_t(e^i_{j,j'}) \tag{3}$$

2) ACTION DESCRIPTION

An action (a_t) is a viable embedding mechanism representing the agent's decisions on embedding VNRs onto the SN resources. The learning agent in *InDS* concentrates on a single VM from a specific $\mathbb{G}_i$ at each step, producing a specific substrate server for embedding. In this work, we follow a coordinated one-stage VNE mechanism. After embedding a VM, a VL embedding is carried out using the BFS shortest algorithm between the servers on which its adjacent VMs are already embedded. The action space in *InDS* is captured in $\mathbb{A} = \{a_1^{1,1}, a_2^{1,2}, \ldots, a_k^{i,j}\}$, where $a_k^{i,j}$ represents the action of assigning VM $v_{i,j}$ to a server s_k. It should be noted that if the embedded VM is the first VM in the specified VNR, there is no VL embedding to be handled for that VM. The action space encloses all potential combinations of actions, such as selecting the servers, allocating resources, and deciding about accepting/rejecting the VNR, subjected to the constraints discussed in Section IV.

3) REWARD DEFINITION

In *InDS*, the learning agent increases the performance by continuously getting a reward (r_t) from the input environment. The reward function informs the agent how well the current action performs compared to other previous actions. To optimize the assessment of overlooked accumulative rewards $E\left(\sum_{t=0}^{\infty} \gamma_t r_t\right)$, the agent might give up the action with the best present reward to achieve better long-term performance. The discount factor γ_t holds at time slot t. A successful action is typically considered acceptable and delivers a positive reward to reinforce the possibility that the present action is established for the embedding process. Otherwise, a failed action will get a negative reward, allowing the agent to explore alternate decisions. The reward function helps in optimizing the VNE problem. The detailed reward computation is discussed in Section V-E.

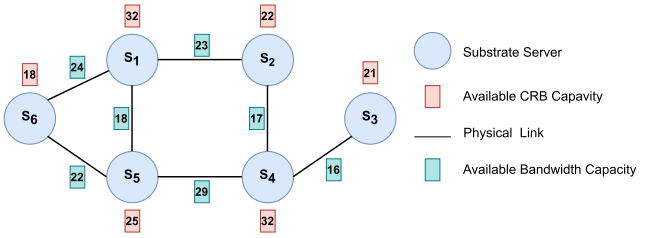

FIGURE 3. An instance of an SN with six servers and seven links and its available resource capacity.

a: LEARNING AGENT

It generates policies based on hybrid GCN and DRL using the network and system-specific raw inputs. The GCNs authorize the DRL agent to learn the network structure and resource distribution in depth. The customized GCN model from the Convolutional Neural Network (CNN) is used to capture these features to cater to maximum VNRs by effectively utilizing the resource to maximize the SP's profit. To achieve this, the agent should competently use input features, cycle through multiple states and actions, and imperatively refine the policy. These policies represent the probability distribution of actions. The detailed learning agent process is described in Section V.

IV. VNE PROBLEM FORMULATION

This section provides a detailed mathematical formulation of the constraints and objectives associated with *InDS*.

A. RESTRICTIONS ON NODE AND LINK EMBEDDING

This section details various node and link embedding restrictions on the assignment of the VM $v_{i,j} \in \mathbb{G}_i$ on the physical server $s_k \in \mathbb{G}_s$ and VL $e^i_{j,j'} \in \mathbb{L}_i$ over a substrate path $e_l \in p_{s_k,s_{k'}}$ over the SN $\mathbb{G}_p$.

1) NODE AND LINK RESOURCE RESTRICTION

Node resource restriction ensures that the server CRB capacity $a(s_k)$ must satisfy the CRB demand $\overline{d}_t(v_{i,j})$ of the VM $v_{i,j}$ being embedded and same is captured as below:

$$\overline{d}_t(v_{i,j}) \leq a(s_k) \tag{4}$$

Similar to node restriction, the link restriction infers that each VL $e^i_{j,j'} \in \mathbb{L}_i$ is associated with the minimum bandwidth demand $\overline{d}_t(e^i_{j,j'})$ required to ensure reliable and efficient communication between VMs $v_{i,j}$ and $v_{i,j'}$ which is embedded over a server s_k and $s_{k'}$, respectively. A link embedding is successful, *iff*, VL demand $\overline{d}_t(e^i_{j,j'})$ is satisfied by the physical link $e_l \in p_{s_k,s_{k'}}$ throughout the path $p_{s_k,s_{k'}}$ over the SN, which is represented as below:

$$\overline{d}_t(e^i_{j,j'}) \leq a(e_l); \ \forall e_l \in p_{s_k,s_{k'}} \tag{5}$$

2) NODE AND LINK INDICATOR VARIABLE

A node indicator variable $\chi(v_{i,j}, s_k)$ is set to 1 if a VM is embedded on an identified physical server. Otherwise, it is assigned 0. Similarly, when a VL is established between two VMs $v_{i,j}$ and $v_{i,j'}$, a link indicator variable $\chi(e^i_{j,j'}, p_{s_k,s_{k'}})$ is set to 1, indicating that the appropriate physical links between the underlying servers s_k and $s_{k'}$ and corresponding indicator variable are initialized as $\chi(v_{i,j}, s_k) = 1$ and the $\chi(v_{i,j'}, s_{k'}) = 1$, respectively. These variables allow for the modeling of communication channels between VMs within VNRs. The same is captured in (6) and (7), respectively.

$$\chi(v_{i,j}, s_k) = \begin{cases} 1 & \text{If } v_{i,j} \text{ is mapped to } s_k \\ 0 & \text{Else} \end{cases} \tag{6}$$

$$\chi(e^i_{j,j'}, p_{s_k,s_{k'}}) = \begin{cases} 1 & \text{If } a(e_l) \geq \overline{d}_t(e^i_{j,j'}), \ \forall e_l \in p_{s_k,s_{k'}} \\ 0 & \text{otherwise} \end{cases} \tag{7}$$

3) NODE MAPPING CONSTRAINT

The node mapping constraint is imposed on the assignment of VMs onto SN servers. It indicates not embedding more than two VMs $v_{i,j}$ and $v_{i,j'}$ of a VNR $\mathbb{G}_i$, on the same server [10], [30], and constraint is represented in (8). The

overall purpose of adding this constraint is to assist SPs in providing distributed services while preventing single-point failures.

$$\chi(v_{i,j}, s_k) \wedge \chi(v_{i,j'}, s_k) \neq 1 \tag{8}$$

4) SUCCESSFUL MAPPING CONSTRAINT

In *InDS*, a VNR is regarded as successfully embedded if both its components, VMs and VLs, are completely allotted to corresponding physical substrate resources, ensuring that resources and connectivity requirements are met. The same is captured in (9) and (10), respectively. The successful embedding signifies the fulfillment of both VM and VL embedding criteria for a given VNR on the SN.

$$|\mathbb{N}_i| = \sum_{\forall v_{i,j} \in \mathbb{N}_i} \min\left(1, \sum_{\forall s_k \in \mathbb{N}_s} \chi(v_{i,j}, s_k)\right) \tag{9}$$

$$|\mathbb{L}_i| = \sum_{\forall e^i_{j,j'} \in \mathbb{L}_i} \min\left(1, \sum_{p_{s_k,s_{k'}} \in \mathbb{P}_{s_k,s_{k'}}} \chi(e^i_{j,j'}, p_{s_k,s_{k'}})\right) \tag{10}$$

5) FLOW CONSERVATION RESTRICTION

The flow conservation restriction confirms that the amount of data traffic entering a node equals the amount of data traffic exiting the node. This restriction is critical for optimal resource allocation and network efficiency. It ensures that data flow through the VL included in the SN is appropriately handled, allowing data packets to be sent efficiently and without interruption. By conforming to this requirement, VNE algorithms can effectively allocate resources [13].

This assertion is given in the (11). In this context, $f^{e^i_{j,j'}}_{s_k,s_{k'}}$ represents the traffic flow of the VL $e^i_{j,j'}$ embedded across the substrate path between servers s_k and $s_{k'}$, which serve as the source and destination servers for the VMs $v_{i,j}$ and $v_{i,j'}$, respectively. The flow into and out of an intermediate server $s_{k''}$ is equal, and $\overline{d}_t(e^i_{j,j'})$ captures the demand of VL $e^i_{j,j'}$ of VNR $\mathcal{G}_i$.

$$f^{e^i_{j,j'}}_{s_k,s_{k'}} - f^{e^i_{j,j'}}_{s_{k'},s_k} = \begin{cases} \overline{d}_t(e^i_{j,j'}), & \text{if } s_{k''} = s_k \\ -\overline{d}_t(e^i_{j,j'}), & \text{if } s_{k''} = s_{k'} \\ 0, & \text{if } s_{k''} \neq s_k, s_{k''} \neq s_{k'} \end{cases} \tag{11}$$

B. EVALUATION OBJECTIVES OF InDS

InDS aims to maximize the revenue-to-cost ratio, improving the acceptance ratio and congestion avoidance by utilizing SN resources to attain more profit for the SPs.

1) ENHANCING THE REVENUE-TO-COST RATIO

The revenue-to-cost ratio is the proportion of the total VM and VL resources demanded by the VNRs to the total substrate resources utilized to cater to these demands. Revenue $\mathbb{R}(\mathbb{G}_i)$ is related to the actual demands of the VNR components defined in (12), whereas (13) describes costs $\mathbb{C}(\mathbb{G}_i)$ as the total physical resources consumed to meet VNR's demand.

$$\mathbb{R}(\mathbb{G}_i) = \sum_{\forall v_{i,j} \in \mathbb{N}_i} \overline{d}_t(v_{i,j}) + \sum_{\forall e^i_{j,j'} \in \mathbb{L}_i} \overline{d}_t(e^i_{j,j'}) \tag{12}$$

$$\mathbb{C}(\mathbb{G}_i) = \sum_{\forall v_{i,j} \in \mathbb{N}_i} \overline{d}_t(v_{i,j}) + \sum_{\forall e^i_{j,j'} \in \mathbb{L}_i} \sum_{\forall e_l \in p_{s_k,s_{k'}}} \overline{d}_t(e^i_{j,j'}) \tag{13}$$

In the meantime, the revenue-to-cost ratio for mapping the request $\mathbb{G}_i$ is computed as follows:

$$\Delta_i = \frac{\mathbb{R}(\mathbb{G}_i)}{\mathbb{C}(\mathbb{G}_i)} \tag{14}$$

2) ENHANCING THE ACCEPTANCE RATIO

The acceptance ratio is the proportion of the number of VNRs successfully embedded to the total number of requested VNRs, which is captured in Equation (15). In which, $\mathbb{A}_{ac}$ and $\mathbb{A}_{ar}$ capture the successfully embedded and total number of arrived VNRs, respectively.

$$\mathbb{A} = \frac{\mathbb{A}_{ac}}{\mathbb{A}_{ar}} \tag{15}$$

3) MINIMIZING THE CONGESTION RATIO

Network congestion occurs when an incoming VNR bandwidth traffic magnitude strikes or exceeds its capacity of SN resources. The exceeding traffic reduces the SN performance and functionality of the embedded VNRs. The proposed *InDS* works on a flow basis to prevent congestion by lowering the congestion ratio $\mathbb{C}_r$ as captured in (16). The VL $e^i_{j,j'} \in \mathbb{L}_i$ between the VMs $v_{i,j}$ and $v_{i,j'}$ which are embedded on servers s_k and s'_k over a substrate link e_l. *InDS* uses this to optimize traffic flow throughout the network, consequently boosting essential performance indicators such as effective resource utilization [31].

The congestion ratio $\mathbb{C}_r$ limits substrate link usage to reduce maximum utilization. The top limit $\mathtt{R}$ is calculated based on the hose traffic demand model [12]. It indicates the bandwidth allocated to VLs on the substrate link e_l over the substrate path $p_{s_k,s_{k'}}$ is limited to the substrate link bandwidth multiplied by the $\mathbb{C}_r$ [12], [24] and the same is captured in (17). The notion $f^{e^i_{j,j'}}_{s_k,s_{k'}}$ represents the flow of traffic demand $\overline{d}_t(e^i_{j,j'})$ from server s_k to server $s_{k'}$ of the embedded VL $e^i_{j,j'}$, i.e., $f(.)$ captures the incoming and outgoing traffic flow from the end servers.

$$C_r = \sum_{e^i_{j,j'} \in \mathbb{L}_i} \frac{f^{e^i_{j,j'}}_{s_k,s_{k'}} + f^{e^i_{j,j'}}_{s_{k''},s_k}}{\overline{a}(e_l)} \tag{16}$$

The value of $\mathtt{R}$ is $0 \leq \mathtt{R} \leq 1$. The chosen value $\mathtt{R}$ is near its upper bound and does not impact the acceptance ratio. We chose $\mathtt{R} = 0.945$ to avoid exceeding the link bandwidth capacity, resulting in unbiased test findings. Using $\mathtt{R}$ as the best congestion ratio ensures it never exceeds the value one,

i.e., $\mathtt{R} < 1$ [12], and it holds $0 \leq \mathbb{C}_r \leq \mathtt{R}$.

$$\sum_{e^i_{j,j'} \in \mathbb{L}_i} (f^{e^i_{j,j'}}_{s_k,s_{k'}} + f^{e^i_{j,j'}}_{s_{k''},s_k}) \leq \mathbb{C}_r * \overline{a}(e_l) \forall\, p_{s_k,s_{k'}} \in \mathbb{L}_s \tag{17}$$

The $v_{i,j}$ and $v_{i,j'}$ are the VMs belong to VNR $\mathbb{G}_i$ embed over a link $e_l \in p_{s_k,s_{k'}}$ and $\overline{a}(e_l)$ denotes the residual substrate link capacity after embedding the VLs of a specific VNR.

Eventually, the comprehensive, objective function of the *InDS* is confined in (18a), (18b) and (18c) are subjected to the following constraints. Constraints (18d) and(18e) capture node CRB and link bandwidth resource restrictions. The node and link indicator variable indicates that a particular VM and VL of a VNR are embedded successfully over the server and link. Same is captured in constraints (18f) and (18g), respectively. Constraint (18h) holds the flow restriction between the two servers. Constraints (18j) and (18i) hold that the congestion ratio over the link cannot exceed the upper limit. Successful mapping of all the VMs and VLs is captured in the constraints (18k) and (18l). The node assignment constraint prohibits assigning two VMs of the same VNR to the same server, and the same is denoted as a constraint (18m). Finally, the possible set of choice variable values is collected in constraints (18n) and (18o).

$$\textit{Maximize} \quad \sum_{\forall \mathbb{G}_i \in \mathbb{G}_v} \mathbb{A}_r \tag{18a}$$

$$\textit{Maximize} \quad \sum_{\forall \mathbb{G}_i \in \mathbb{G}_v} \Gamma_i \tag{18b}$$

$$\textit{Minimize} \quad \sum_{\forall e_l \in \mathbb{L}_s} \mathbb{C}_r \tag{18c}$$

$$\text{s.t.} \quad \overline{d}_t(v_{i,j}) \leq a(s_k) \tag{18d}$$

$$\overline{d}_t(e^i_{j,j'}) \leq a(e_l); \;\; \forall e_l \in p_{s_k,s_{k'}} \tag{18e}$$

$$\sum_{\forall s_k \in \mathbb{N}_s} \chi(v_{i,j}, s_k) = 1 \tag{18f}$$

$$\sum_{\forall e_l \in p_{s_k,s_{k'}}} \chi(e^i_{j,j'}, p_{s_k,s_{k'}}) = 1 \tag{18g}$$

$$f^{e^i_{j,j'}}_{s_k,s_{k'}} - f^{e^i_{j,j'}}_{s_{k'},s_k} = \begin{cases} \overline{d}_t(e^i_{j,j'}), & \text{if } s_{k''} = s_k \\ -\overline{d}_t(e^i_{j,j'}), & \text{if } s_{k''} = s_{k'} \\ 0, & \text{if } s_{k''} \neq s_k,\, s_{k''} \neq s_{k'} \end{cases} \tag{18h}$$

$$\sum_{e^i_{j,j'} \in \mathbb{L}_i} (f^{e^i_{j,j'}}_{s_k,s_{k'}} + f^{e^i_{j,j'}}_{s_{k''},s_k}) \leq \mathbb{C}_r * \overline{a}(e_l) \forall\, p_{s_k,s_{k'}} \in \mathbb{L}_s \tag{18i}$$

$$0 \leq \mathbb{C}_r \leq R \tag{18j}$$

$$|\mathbb{N}_i| = \sum_{\forall v_{i,j} \in \mathbb{N}_i} \min \left(1, \sum_{\forall s_k \in \mathbb{N}_s} \chi(v_{i,j}, s_k)\right) \tag{18k}$$

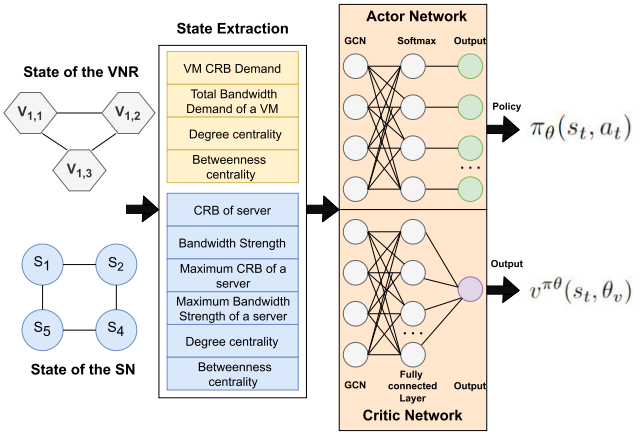

FIGURE 4. An instance of a learning agent process used in *InDS*.

$$|\mathbb{L}_i| = \sum_{\forall e^i_{j,j'} \in \mathbb{L}_i} \min \left(1, \sum_{p_{s_k,s_{k'}} \in \mathbb{P}_{s_k,s_{k'}}} \chi(e^i_{j,j'}, p_{s_k,s_{k'}})\right) \tag{18l}$$

$$\chi(v_{i,j}, s_k) \wedge \chi(v_{i,j'}, s_k) \neq 1 \tag{18m}$$

$$\forall s_k \in \mathbb{N}_s; \quad \forall e^i_{j,j'} \in \mathbb{L}_i; \quad \forall v_{i,j}, v_{i,j'} \in \mathbb{N}_i \tag{18n}$$

$$\forall p_{s_k,s_{k'}} \in \mathbb{P}_{s_k,s_{k'}}; \quad \forall e_l \in p_{s_k,s_{k'}}; \quad \forall\, \mathbb{G}_i \in \mathbb{G}_v \tag{18o}$$

V. SOLUTION APPROACH

This section provides detailed functionalities of the *InDS* learning agent as demonstrated in Figure 4. The learning agent produces appropriate policies based on the extracted features. The approximation-based neural network with trainable parameters extracts the features and generates policies from environmental input states. In order to train these input attributes, a policy gradient embedding strategy is used to enhance the proposed embedding policy [32]. The subsequent sections further explain the feature extraction, policy development, and training strategy. Note that the VMs and servers are referred to as nodes throughout the subsequent sections.

A. FEATURE EXTRACTION

The GCN primarily uses Laplacian matrix ($\mathtt{L}$) and orthogonal factorization ($\mathtt{U}$) to describe the SN and VNRs features during the processing of the requests (refer to Section III-B). It involves a Fourier transformation in an n-dimensional space, resembling the traditional Fourier transform that decomposes a real-value function into orthogonal functions. It behaves as a semi-supervised learning for convolution operation on random graph topology based on spectro-graph theory [14], [16]. The corresponding Fourier transform is described as below:

$$F(\lambda_v) = \widehat{f}(\lambda_v) = \sum_{i=1}^{|\mathbb{N}|} f(i)\mu^*_v(i) \tag{19}$$

In which, $f(i)$ represents the network node, $\mu_v^*(i)$ conveys the i^{th} component of the v^{th} feature vector, λ_v is the eigen value. The Fourier transformation of the input vector f with dimension $|\mathbb{N}|$, represented by $\widehat{f}$, on network $\mathbb{G}_v$ or $\mathbb{G}_s$, is computed using (20), where $\mathtt{U}=\{u_1, u_2, .., u_{n-1}\}$ represents Fourier basis.

$$\widehat{f} = \mathtt{U}^T f \tag{20}$$

The Fourier transform of a convolution of two functions is the pointwise product of their Fourier transforms [33]. Then, the Fourier transform of f and convolution kernel signal y on the network topology is represented as follows:

$$f * y = \Gamma^{-1}(\widehat{f}(w).\widehat{y}(w)) = \frac{1}{2\pi}\int \widehat{f}(w).\widehat{y}(w)e^{-iwt}dw \tag{21}$$

The $y = \sum_{k=0}^{K} \alpha_k \Lambda^k$ holds the kernel filter utilized for finer feature extraction. In which the trainable characteristics are seized in α_k and Λ^k represents the diagonal matrix with eigenvalues of $\mathtt{L}_k$. Finally, (22) provides the result of the GCN model, where σ represents the neural network's activation function, and the order index K specifies locality.

$$y_{out} = \sigma(\sum_{k=0}^{K} \alpha_k \mathtt{L}_k f) \tag{22}$$

A network node's recognition field can reach neighboring nodes within K hops under the specified kernel. Features from the SN and VNRs are extracted using two single-layer GCNs. The feature matrices corresponding to the SN and VNRs are processed for every state. It provides improved node representations, containing crucial structural and relational data about the two networks, by transmitting these matrices across the GCN layer. Each node is represented as a vector with the specified hidden dimensions by the GCN, which creates a tensor of size equal to the product of the number of nodes in a network and hidden units, and hidden units are set to eight in *InDS*. The six features extracted from the SN are combined into one vector. Similarly, the VNR creates an embedding by combining four features.

B. POLICY GENERATION

After the GCN extracts the features from both the substrate and the VNRs, these features are combined and sent via a fully linked layer to learn complex relationships. In an RL system, a policy layer generates a probability distribution over substrate servers for each VM, which are then converted into action probabilities by a softmax layer for efficient VM embedding [34]. The softmax function turns a neural network's output into a probability distribution for diverse embedding actions. This distribution aids decision-making by giving each action a_t probability, allowing for the testing of various embedding strategies. During training, the policy network uses DRL to optimize predicted rewards, often measured by resource consumption and network attribute status. Once trained, the policy network develops embedding policies in real-time, allowing for adaptive and efficient VNE solutions for changing network conditions.

The input layer calculates the feature matrix of the server and passes it to the convolutional layer. This layer convolves the matrices, producing a vector reflecting each server's available resources as in (23). The variable h_k indicates the softmax's k^{th} output with the weight vector w_k, b_k holds the bias, and V_k captures the k^{th} feature vector. The softmax layer converts k into the likelihood of each server being selected. A high-probability server is then chosen to achieve a superior embedding outcome.

$$h_k = \begin{cases} w_k * V_k + b_k & \text{if } w_k * V_k + b_k > 0 \\ 0 & Otherwise \end{cases} \tag{23}$$

C. ACTOR CRITIC BASED POLICY GRADIENT TRAINING

The current work *InDS* uses the A3C model for training the agent [35], [36]. In A3C, the actor network responsible for generating the embedding policy $\pi_\theta(s_t, a_t)$ and the critic network produces the values $\mathtt{V}^{\pi_\theta}(s_t, \theta_v)$. This approach uses simultaneous threads to train the *InDS* model. Multiple agents operate concurrently in the local network, interacting with the environment independently. At the start of each episode, each agent receives a copy of the global network settings. After sampling their experiences, the agent uses the loss functions to change the parameters of its local actor-critic network. The mean squared error loss for the critic is computed by $\sum[\mathtt{V}(s_t) - \mathtt{G}^2]$. The $\mathtt{V}(s_t)$ is the predicted state-value or state-action value by the critic, and $\mathtt{G}$ is the observed cumulative reward obtained by the agent after taking an action in the state s_t. Similarly, the policy loss is defined $-\log\left(\prod(a_t|s_t)\right) * \mathtt{A}$. The expression $\prod(a_t|s_t)$ represents the probability assigned by the actor to selecting action a_t given the state s_t. At the end of each agent's episode, the global network receives asynchronous updates about its learning from the local network. In the actor-critic strategy, the critic provides feedback to the actor by transmitting a gradient known as the advantage or temporal-difference (TD) error gradient [37]. The TD error can be used to train the neural network by altering its parameters to minimize the error. The difference between predicted and actual observed rewards is used by TD learning to update value estimations across succeeding time steps. This strategy balances the bias-variance trade-off by continuously changing value estimates as fresh data is observed. This reduces the possibility of overestimation while improving training process stability [38]. Back-propagation calculates the gradient of the TD error concerning neural network parameters, which is used to adjust the network's weights using stochastic gradient descent. Backpropagation-based sensitivity analysis involves calculating the output gradient concerning each input feature. This helps understand the influence of each input feature on the output. During training, backpropagation calculates the temporal difference error gradient concerning the neural network parameters, adjusting the network's weights using stochastic gradient

descent. This process fine-tunes the weights and highlights the importance of individual features based on their gradients.

This gradient is an essential component of the policy gradient approach for updating the policy-based actor network. The advantage gradient shows that the selected action is better or worse than the predicted value at a given stage. It reflects the advantage of choosing a specific action above other options accessible in that state, depending on the critic's current value function estimates. The advantage function helps address the gap between the expected reward for a chosen action a_t and the intermediate state significance calculated as described below:

$$A^{\pi_\theta}(s_t, a_t) = Q^{\pi_\theta}(s_t, a_t) - \mathbb{V}^{\pi_\theta}(s_t) \tag{24}$$

The state value function that determines the accumulative return in state s_t is denoted by $\mathbb{V}^{\pi_\theta}(s_t)$. For a given state s_t, the advantage function shows the relative improvement of the current action above the average action obtained from the associated policy. It is possible to lower the variance during training without changing the bias. In the meantime, there is a more significant distinction between actions taken in the same condition. In specifically, for a disseminated experience (s_t, a_t, r_t, s_{t+1}), the estimation of $Q^{\pi_\theta}(s_t, a_t)$ is determined by $r_t + \gamma_t \mathbb{V}^{\pi_\theta}(s_t + 1)$. Now $A^{\pi_\theta}(s_t, a_t)$ is later estimated as $r_t + \gamma_t \mathbb{V}^{\pi_\theta}(s_t + 1) - \mathbb{V}^{\pi_\theta}(s_t)$. Therefore, (25) states that the actor-network update with the aid of the advantage function follows the policy gradient training.

$$\theta = \theta + \alpha \sum_t \nabla_\theta \log \pi_\theta(s_t, a_t) A^{\pi_\theta}(s_t, a_t) + \beta \nabla_\theta H(\pi(.|s_t)) \tag{25}$$

The α represents the learning rate used in *InDS*, and the policy for all actions in the current state s_t is captured in $\pi_\theta(.|s_t)$. The actor parameter set θ gradient directs adjustments to network parameters, impacting the probability $\pi_\theta(s_t, a_t)$. Similarly, θ_v denotes the critic parameter set. Over successive training episodes, the agent reinforces actions with superior empirical rewards in specific states. The entropy term $H(.)$ promotes a more uniform action distribution, acting as regularization and limiting premature convergence to local optima with decaying attribute $\beta = 0.5$. The critic network learns using the TD technique, which is defined as:

$$\theta_v = \theta_v + \alpha' \sum_t \nabla \theta_v (r_t + \gamma_t \mathbb{V}^{\pi_\theta}(s_{t+1}; \theta_v) - \mathbb{V}^{\pi_\theta}(s_t; \theta_v))^2 \tag{26}$$

The *InDS* has been effectively scaled by enforcing a parallel training framework where four agents operate concurrently, each utilizing a shared set of actor-critic network parameters derived from a central agent. These agents collectively enhance learning efficiency by gathering diverse experiences simultaneously, thereby mitigating the bottleneck associated with sequential experience accumulation. The central agent serves as a repository for network parameters, continuously updating them based on aggregated experiences from all agents. Upon completion of each training episode, updated parameters are disseminated to all agents, ensuring synchronous progress and parameter alignment across the system. This approach accelerates the training process. To address the termination condition, the training process is bounded by a predefined maximum number of episodes, i.e., 5000, ensuring that the agent does not undergo infinite training cycles. This strategic capping facilitates a balanced trade-off between exhaustive learning and computational efficiency, promoting a robust and scalable DRL model. The central agent uses the experiences from independent sources to mitigate the correlation among state-action pairs within trajectories. After training completion, the model is subjected to testing using the parameters detailed in Table 3. The subsequent section elaborates on the end-to-end embedding strategy.

D. TESTING PHASE

The Algorithm 1 demonstrates the step-by-step procedure involved in *InDS* to perform node and link embedding procedure. It takes PN $\mathbb{G}_p$ and VNRs $\mathbb{G}_v$ as input during the testing process. Initially, Step (1) of Algorithm 1 performs the initialization of all the VMs to free, and none of the VMs are rejected. All the VNRs $\mathbb{G}_i \in \mathbb{G}_v$ are set to free from the Steps (2-4) of Algorithm 1. Select the VNR $\mathbb{G}_i$ from the $\mathbb{G}_v$ based on arrival order. The VM $v_{i,j} \in \mathbb{N}_i$ is chosen based on the sequence that is present during the generation, and initially, the number of attempts is set to zero from Steps (5-8) of Algorithm 1. Then, from Step (9) of Algorithm 1 compute the probability value of each server using *ActorNetwork* (state, $v_{i,j}$) and stored in the $\mathbb{P}(.)$. Followed by this, a server with the highest probability value is selected for VM embedding adhering to CRB constraints, and PN resources are updated accordingly from the Steps (10-19) of the Algorithm 1, and embedded VM is added to the set $\mathbb{N}'_i$. Otherwise, the next high-probability server is selected, and suitable action will be performed from the Steps (20-29) of the Algorithm 1. If VM embedding fails for a specific VNR, then such VNRs are set to F and release the allocated substrate resources from the Steps (30-35) of the Algorithm 1. In the case of successful VM embedding, a VL embedding is carried out between the current VM and already embedded VMs in $\mathbb{N}'_i$ using the shortest path algorithm from Steps (37-41) of the Algorithm1. Otherwise, such VNR is rejected, and already allocated resources will be released from the Steps (42-48) of the Algorithm 1. The above steps will be repeated for all VNRs in $\mathbb{G}_v$, and finally, mapping results are captured in the set $\mathbb{M}$ of Step (52) of the Algorithm1. The following section provides the detailed experimental results captured during the testing phase.

E. REWARDING IN InDS

The RL is unsupervised, and the training set contains no labels. In the policy gradient approach, the RL agents change policies depending on the reward signals. *InDS* employs a customized reward process tailored to specific assignments.

Algorithm 1 *InDS* Embedding Strateg

Input: $\mathbb{G}_v$, $\mathbb{G}_p$
Result: $\mathbb{M} : \mathbb{V} \rightarrow \mathbb{N}_p$
Initialization: $free[v_{i,j}] = T$, $reject[v_{i,j}] = F$
for *each* $\mathbb{G}_i \in \mathbb{G}_v$ **do**
end
$free[\mathbb{G}_i] = T$
while $\mathbb{G}_v \neq \Phi$ **do**
  while $\exists\, v_{i,j} \in \mathbb{N}_i \mid (free[v_{i,j}]\ and\ !\, reject[v_{i,j}])$ **do**
    $v_{i,j}$ = **Pick** ($\mathbb{N}_i$)
    No_Attempts = 0 ▷ The number of allocation attempts.
    $\mathbb{P}$ = **ActorNetwork**(state, $v_{i,j}$)
    while $\mathbb{N}_p \neq \Phi$ *and* $(free[v_{i,j}]\ and\ !\, reject[v_{i,j}])$ **do**
      s_k = **HighprobSelect**($\mathbb{P}$)
      No_Attempts++
      if $a(s_k) \geq \overline{d}_t(v_{i,j})$ **then**
        free[$v_{i,j}$] = F
        $\mathbb{N}_p = \mathbb{N}_p \setminus \{s_k\}$
        $\mathbb{M} = \mathbb{M} \cup \{(v_{i,j}, s_k)\}$
        $a(s_k) = a(s_k) - \overline{d}_t(v_{i,j})$
        $\mathbb{N}'_i = v_{i,j}$
      end
      else if $No\_Attempts < |\mathbb{N}_p|$ **then**
        s_k = **NextHighprobSelect**($\mathbb{N}_p$)
        No_Attempts++
        if $a(s_k) \geq \overline{d}_t(v_{i,j})$ **then**
          free[$v_{i,j}$] = F
          $\mathbb{N}_p = \mathbb{N}_p \setminus \{s_k\}$
          $\mathbb{M} = \mathbb{M} \cup \{(v_{i,j}, s_k)\}$
          $a(s_k) = a(s_k) - \overline{d}_t(v_{i,j})$
          $\mathbb{N}'_i = v_{i,j}$
        end
      end
      else
        reject[$v_{i,j}$] = T
        free[$\mathbb{G}_i$] = F
        Release_PN_Resources ($\mathbb{G}_i$)
        break;
      end
    end
    if $\exists\, v_{i,j} \in \mathbb{N}'_i \mid reject[v_{i,j}] = F$ **then**
      for *each* $(v_{i,j'} \in adj(v_{i,j})\ \&\ v_{i,j'} \in \mathbb{N}'_i) \mid !(AlreadyEmbeded\ (e^i_{j,j'}))$ **do**
        if *Feasible_Path* $(\mathbb{M}(v_{i,j}), \mathbb{M}(v_{i,j'}), r(e^i_{j,j'}))$ **then**
          Reserve_Path $(\mathbb{M}(v_{i,j}), \mathbb{M}(v_{i,j'}), r(e^i_{j,j'}))$
        end
        else
          free[$\mathbb{G}_i$] = F
          Release_PN_Resources ($\mathbb{G}_i$)
          break;
        end
      end
    end
  end
  $\mathbb{G}_v = \mathbb{G}_v \setminus \{\mathbb{G}_i\}$
end
return $\mathbb{M}$

This ensures the agent receives more specific feedback, reducing the potential for overestimating future rewards. The reward will be assigned as a positive or negative reward based on the successful or unsuccessful embedding results. A larger reward signifies a valid action, whereas a smaller or negative reward suggests an invalid activity that requires restoration. *InDS* set an initial reward of $\lambda = 100$ and $\lambda = -100$ for the positive and negative reward respectively. As a result, the acceptance outcome of the reward is captured as below:

$$r_a = \begin{cases} \lambda\gamma_t & action\ is\ success \\ -\lambda\gamma_t & otherwise \end{cases} \tag{27}$$

In which γ_t captures the discount factor, which begins at $1/|\mathbb{N}_v|$ and slowly rises for processing of each VM. Note that the discount factor weight is less for the first VM and gradually increases for the last VM of a specific VNR.

Furthermore, the learning agent must take successful and cost-effective behaviors while processing the VNR. A better embedding policy, such as shorter substrate paths, is possible by *InDS* due to the incorporation of the network features, such as degree and betweenness centrality, along with system resources during the training. As a result, we add another element to the reward function as denoted in (28).

$$r_r = \frac{\delta(\mathbb{R})}{\delta(\mathbb{C})} \tag{28}$$

The variable $\delta(\mathbb{R})$ captures the new revenue, and $\delta(\mathbb{C})$ holds new cost resulting from the current action compared to the previous step. Further, congestion avoidance is another parameter to be met for each action, a_t, and a low congestion ratio is rewarded. The mean of the congestion ratio for every newly embedded link from the current VM along its path is determined as follows:

$$r_c = \frac{r(e_{j,j'})}{a(e_l)} \tag{29}$$

Finally, *InDS* employs the eligibility check $\mathbb{E}^c$ for each action i as captured in (30) to avoid embedding policies that get stuck in the repeated actions frequently.

$$\mathbb{E}^c_t[i] = \begin{cases} d_e(\mathbb{E}^c_{t-1}[i] + 1) & i = a_t \\ d_e(\mathbb{E}^c_{t-1}[i]) & i \neq a_t \end{cases} \tag{30}$$

In *InDS*, the decay factor d_e is set to 0.99, which reduces the eligible trail slightly at each time stamp. This strategy ensures that regularly selected actions receive constant inspiration and keep a high value for an extended period, whereas unpicked actions gradually decay to zero. We use this eligibility check to partition the reward function. It eliminates activities that are not frequently chosen, thereby eliminating the risk of training becoming a sub-optimal state. Finally, the action a_t has the following reward function.

$$Reward[a_t] = \frac{r_a * r_r * r_c}{\mathbb{E}^c_t[i] + \mu} \tag{31}$$

In which μ is a tiny positive number to prevent the denominator from zero.

TABLE 3. Simulation parameters.

Parameters	Value
No. of substrate Server	84
Servers CRB capacity	U[10 − 500]
No. of substrate Link	93
Link Bandwidth capacity	U[10 − 500]
Range of VMs in a VNRs	U[2 − 10]
Bandwidth demand	U[1 − 5]
CRB demand	U[1 − 10]
Probability of VL connection	0.4
Total Test scenarios	4
VNRs per scenarios	[250, 500, 750, 1000]
Number of iterations per scenario	10
Number of Episodes	5000
Number of hidden units	8

VI. PERFORMANCE ANALYSIS AND EVALUATION

This section describes the experimental setup used in *InDS* and provides a detailed analysis of the simulation results using multiple evaluation criteria comparison with diverse baseline works.

A. EXPERIMENTAL SETUP

To evaluate the performance of the *InDS*, the simulation experiment is run over a Windows 64-bit operating system with a Ryzern processor utilizing *alib utility* kit [5], [39], [40]. The *InDS* source code is made freely accessible at [19] and developed using the Python 3.9 version. The *alib utility* kit is a python library incorporating real-time physical network topologies information from *Internet Topology Zoo* [41], [42]. Each network's topology information is saved in a *yml* file based on publicly available data provided by diverse network operators. It is the most accurate collection of large-scale real-time SN topology data, and the metadata given aids in understanding its structure. We simulated an SN with 84 servers and 93 linkages. The cost of unit SN resource usage is charged at 1$ [5], [25]. The simulation parameter settings for training are established according to works [2], [11], [14], [20], and the same is described in Table 3. In addition, we build a diverse VNR set for each scenario, with uniformly generated CRB and bandwidth demands for testing. The majority of real-world network traffic demonstrates random VNR arrival times. The Poisson distribution serves exceptionally well at replicating random behavior and independent requests. We generated VNRs using a Poisson distribution with a mean of $\lambda = 0.4$. The λ value in *InDS* implies a relatively low average arrival rate of 0.4 requests per unit of time [2]. Meanwhile, 25% of VNRs suffer resource upgrades or downgrades for VMs and VLs in the range [1, 10] and [1, 5], respectively, indicating an automatic DRL setup. Figure 5 depicts the distribution of the VMs and VLs independently. Meanwhile, Figures 6 to 14 illustrate the combined performance of initial and remapping VNRs due to resource change, which is elaborated in subsequent sections.

B. THE BASELINE STRATEGIES

To evaluate the usefulness of *InDS*, we compare its performance with four distinct baseline approaches as follows.

- ***VNE-NRM*** [11]. This work presents a heuristic technique for properly managing SN resources by considering various system characteristics. It employs a ranking technique for VMs and servers that considers a variety of computational parameters during VM embedding followed by a shortest path VL embedding. It aims to increase acceptance and revenue-to-cost ratios by making better use of substrate resources.
- ***A3C-GCN*** [14]. The author proposes A3C-GCN, a learning-based VNE strategy that uses reinforcement learning techniques with GCNs. It considers diverse system resource parameters during training using the actor-critic model. This strategy tries to achieve effective load distribution and maximize the revenue-to-cost ratio. ***VNE-MWF*** [6]. In this work, the authors proposed a modified worst-fit embedding strategy considering system attributes. The servers and VMs are selected based on maximum capacity during VM embedding. Further, it uses a shortest-VL embedding to enhance the acceptance and revenue-to-cost ratios.
- ***DPGA*** [20]. The authors developed a GA-based meta-heuristic strategy to address the VNE problem. This approach applies a series of crossovers and mutations during VM embedding. Further, it regulates VL embedding by creating an initial path collection using the BFS shortest path algorithm to improve acceptance and revenue-to-cost ratios.

InDS primary aim is to enhance the revenue-to-cost ratio, and acceptance ratio and minimize network congestion compared to baselines [6], [11], [14], [20]. It enables accurate and fair validation of *InDS* performance enhancements.

C. THE PERFORMANCE ANALYSIS

The performance of *InDS* is tested against diverse baseline techniques using three assessment measures, i.e., (*i.*) Consumption of Physical Resources, (*ii.*) Quality of Service, (*iii.*) Congestion Metrics and (*iv.*) Other relevant Metrics [2], [5], [12].

1) CONSUMPTION OF PHYSICAL RESOURCES

This section describes the various physical resource consumption metrics, such as acceptance and revenue-to-cost ratios.

a: ACCEPTANCE RATIO

It refers to the proportion of successfully embedded VNRs to the total number of incoming VNRs. Figure 6 demonstrates the acceptance ratios of various state-of-the-art techniques, revealing that *InDS* has a superior acceptance ratio across diverse use cases. Figure 6 shows that the average acceptance

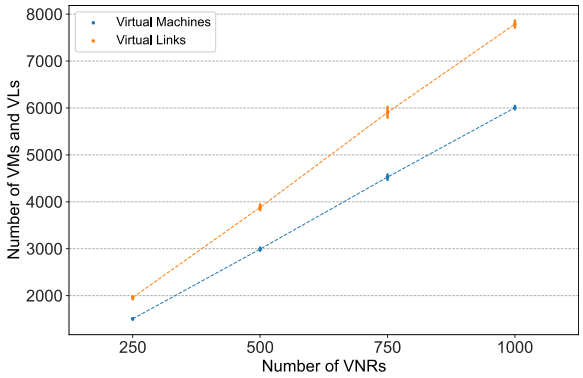

FIGURE 5. Number of VMs and VLs *vs.* Number of VNRs.

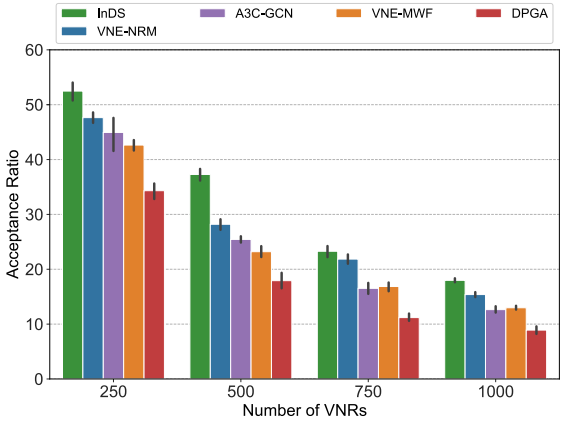

FIGURE 6. Acceptance ratio *vs.* Number of VNRs.

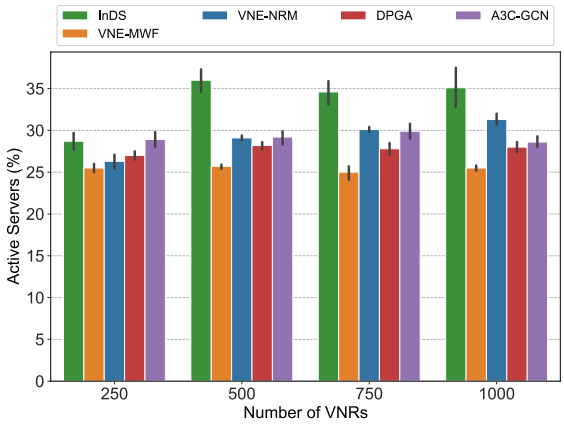

FIGURE 7. Active servers (%) vs. Number of VNRs.

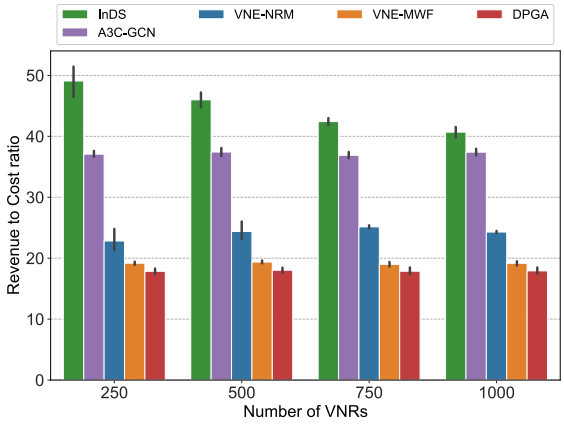

FIGURE 8. Revenue to cost ratio vs. Number of VNRs.

ratio lowers in subsequent cases due to limited SN resources. Integrating network and system-specific features in *InDS* during training accounts for its improved performance compared to baseline approaches such as VNE-NRM, A3C-GCN, VNE-MWF, and DPGA. Incorporating continuous learning utilizing DRL from several episodes and extracting the diverse attributes employing GCNs is a fundamental cause for the enhanced acceptance ratio. Further, *InDS* uses more active servers than all other strategies (refer to Figure 7). This is due to the acceptance of more VNRs than different strategies. Meanwhile, VNE-NRM performs inferior to *InDS* due to a poor embedding mechanism based on only system resource-specific ranking, resulting in inefficient resource usage, which is the reason for its inferior performance. On the other hand, A3C-GCN results in poorer substrate resource utilization due to neglecting network-specific features during training, which leads to a lower acceptance ratio than VNE-NRM and *InDS*. Further, the mechanism used in VNE-MWF's assignment leads to more distributed VM placement, resulting in lower performance. DPGA uses randomness in mutation and cross-over operation, which is the reason for its decaying performance compared to all other approaches and *InDS*.

b: REVENUE TO COST RATIO

It is the ratio of used resources to total available substrate resources for successfully embedding the VNRs. Figure 8 displays the average revenue-to-cost ratio across the baselines. The proposed *InDS* exhibits a superior revenue-to-cost ratio than the VNE-NRM, VNE-MWF, A3C-GCN, and DPGA baselines. *InDS* considering network features along with system resources during the training phase results in proximal VM embedding, which results in few substrate link usages. This helps *InDS* accommodate more VNRs, thereby enhancing the revenue-to-cost ratio in the long run. The negligence of the network features by A3C-GCN is the reason for its decayed performance than *InDS*. Further, VNE-NRM uses a ranking mechanism that leads to more scattered VM placement, resulting in poorer performance than the A3C-GCN and *InDS*. On the other hand, the VNE-MWF conducts the VM embedding established on maximum available server capacity, which results in more dispersed VM placement, leading to more substrate path consumption for fewer VNRs, which is the reason for its decreased performance than *InDS*. Further, DPGA shows the least performance compared to all other baselines and *InDS* due to randomness in its solution.

2) QUALITY OF SERVICE (QoS)

This section explains the diverse QoS metrics, such as server utilization, link utilization, and substrate path length utilization.

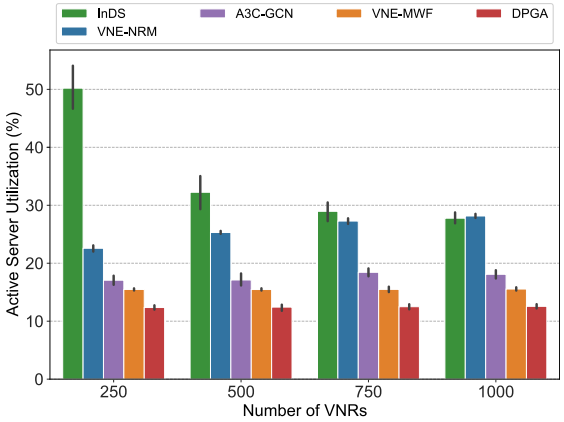

FIGURE 9. Active server utilization (%) vs. Number of VNRs.

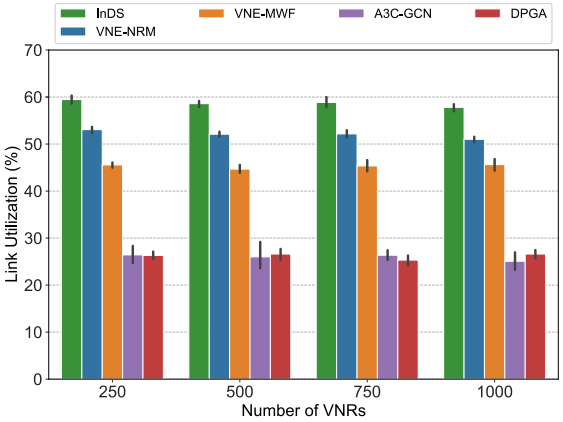

FIGURE 10. Link utilization (%) vs. Number of VNRs.

a: SERVER UTILIZATION

It determines the percentage of the server's CRB resource capacity used by the embedded VMs to the overall available server's CRB resource capacity. Figure 9 indicates the average server resource consumption across the baselines. The proposed *InDS* consumes more server resources for initial test cases and decreases for preceding larger test cases due to decreased acceptance compared to other baseline strategies. This behavior of NORD attributed to considering multiple features during training results in better embedding decisions, which helps in maximizing server utilization by accommodating more VNRs. Further, VNE-NRM devastates lower substrate resource utilization than *InDS*. However, it is superior to A3C-GCN, VNE-MWF, and DPGA. A3C-GCN exhibits poorer performance than VNE-NRM and *InDS* due to a lack of sophisticated network-related features, which causes dispersed VM placement and leads to lower substrate resource utilization. Similarly, VNE-MWF adopts a poor embedding strategy, resulting in a lack of network resources for accepting more VNRs. This leads to poor server resource utilization over the test scenarios than *InDS*. On the other hand, due to the involvement of random operations, DPGA exhibits inferior resource utilization than all other strategies.

b: LINK UTILIZATION

It reflects the ratio of consumed bandwidth link resources to the available physical link's capacity. Figure 10 depicts the link resource used for various baseline strategies. The *InDS* uses more link resources than the VNE-NRM, A3C-GCN, DPGA, and the VNE-MWF strategies. This behavior is linked to satisfying maximum utilization limitations on substrate links to avoid congestion (refer to Section IV-B3). VNE-NRM and VNE-MWF use a system resource-specific embedding technique, resulting in poor substrate link utilization than *InDS*. Alternatively, A3C-GCN consumes fewer link resources than the VNE-NRM and *InDS* due to the poor acceptance ratio throughout the test cases. VNE-MWF consumes more link resources than the A3C-GCN due to fairly dispersed VM placement, resulting in more link resource consumption for fewer VNRs. However, the involvement of mutation and crossover in DPGA leads to degraded performance than *InDS* and other approaches.

c: PATH LENGTH UTILIZATION

The average path length measures the efficiency of mapping VNRs onto the SN based on the length of the paths consumed. Due to link upper bound capacity constraints, the proposed *InDS* approach consumes more average substrate paths across the cases, which is captured in Figure 11. This statistic represents congestion prevented by distributing VL requests over more substrate links by *InDS* compared to other baseline techniques such as A3C-GCN, VNE-MWF, VNE-NRM, and DPGA. On the other hand, due to the randomization in the embedding technique, the DPGA uses fewer substrate path lengths than NORD. However, it consumes more substrate path length than the VNE-NRM, A3C-GCN, and VNE-MWF strategies. Alternatively, VNE-NRM and VNE-MWF have lower average path length usage than *InDS* for a lower acceptance ratio due to the system attribute-based embedding mechanism. A3C-GCN uses less average path length than all other techniques because it relies solely on system metrics during learning. However, all baseline techniques overlooked congestion avoidance over links, resulting in reduced performance compared to the *InDS*.

3) CONGESTION METRICS

This section explains the congestion metrics, such as average server and link stress.

a: AVERAGE SERVER STRESS

It quantifies the load on individual servers in a network resulting from embedded VMs from the VNRs. Figure 12 demonstrates the average server stress across the SN. In *InDS* servers are not overloaded, which will improve the overall performance. It implies that *InDS* congestion-control strategy leads to better load distribution across servers than the other baseline approaches by achieving maximum acceptance and

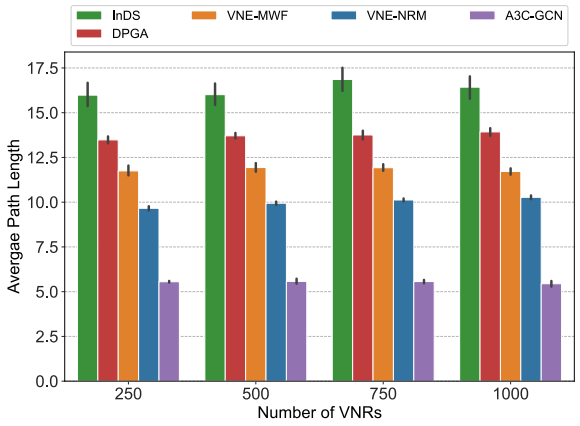

FIGURE 11. Average path length vs. Number of VNRs.

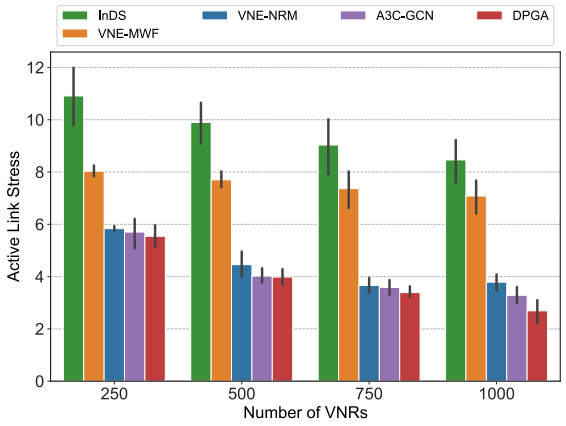

FIGURE 13. Active link stress vs. Number of VNRs.

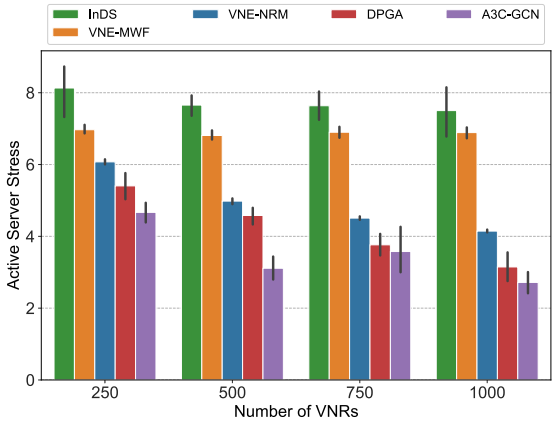

FIGURE 12. Active server stress vs. Number of VNRs.

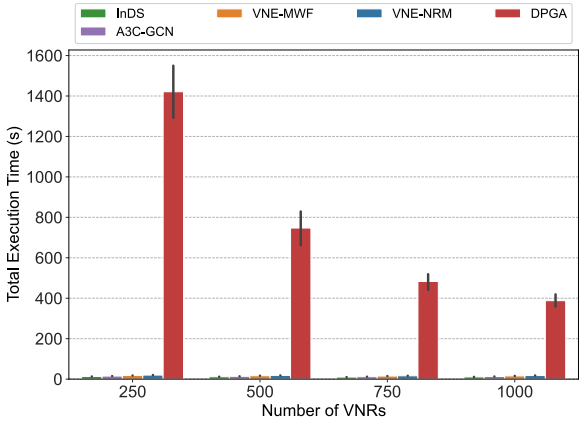

FIGURE 14. Total execution time (s) vs. Number of VNRs.

revenue-to-cost ratios. On the other hand, VNE-MWF uses a kind of ranking based on resource availability during VM embedding, which is superior to A3C-GCN, VNE-NRM, and DPGA approaches across the scenarios. Meanwhile, VNE-NRM shows slightly less server consumption than *InDS* and VNE-MWF, and this is due to the use of a system resource-specific ranking mechanism. On the other hand, A3C-GCN delivers slightly lower server consumption than the VNE-NRM, and *InDS* across the test scenarios due to negligence of the network features during training is the reason for its behavior. The strategy DPGA demonstrates a decreased acceptance ratio is the reason for its inferior performance compared to all other approaches and *InDS*. Moreover, VNE-NRM. VME-MWF and DPGA neglected the load balancing in their strategies, leading to diminished performance.

b: AVERAGE LINK STRESS

It is the proportion of the total number of embedded VLs by the total number of substrate links, and the same is captured in Figure 13. Due to the upper bound limit on the link usage to achieve the congestion avoidance objective, the *InDS* archives superior performance than other approaches. Moreover, considering system and network-specific features during training results in improved acceptance and effective link load distribution. Alternatively, the strategies VNE-NRM, DPGA, and VNE-MWF show almost similar performance due to a poorer acceptance ratio. On the other hand, due to a lack of network features during the training in A3C-GCN resulted in diminished link utilization compared to all other baselines and *InDS*. Moreover, these approaches neglected congestion avoidance, leading to decreased performance than *InDS*.

4) OTHER RELEVANT METRICS

The other relevant metrics reflect the execution time consumed by different strategies.

a: EXECUTION TIME

DPGA takes the longest time to execute all test cases due to advanced crossover and mutation processes, resulting in high execution time overhead. VNE-NRM and VNE-MWF have slightly longer runtime than A3C-GCN and *InDS*. This overhead is due to the use of a ranking process during VM and server selection. On the other hand, both A3C-GCN and *InDS* demonstrate lower execution compared to all other techniques due to intensive training lowering its assignment time during the embedding phase.

From the above illustration, we deduce that *InDS* delivers significant performance increases compared to various baselines employing considerable evaluation criteria.

VII. CONCLUSION AND FUTURE DIRECTIONS

This work proposes a multi-objective framework called *InDS* to address fundamental issues in NV, particularly the VNE. *InDS* uses hybrid DRL and GCN-based techniques to dynamically extract network and system-specific features based on the network environment state during the training. To accelerate the training process and produce a more efficient training experience, we utilized parallel training for policy generation using the A3C algorithm. *InDS* learning agent uses GCN to extract multiple characteristics from raw state inputs efficiently. The recommended reward function efficiently controls the learning process by considering diverse objectives such as congestion avoidance, acceptance, and revenue-to-cost ratio maximization by effectively utilizing the substrate resources. The DRL enhances the decision-making capabilities of *InDS* by considering four key attributes such as CRB, bandwidth, degree, and betweenness centrality. The DRL agent is trained within variants of this four-dimensional network environment, ultimately determining the probability of VNRs being mapped onto specific servers in the SN. The performance evaluation demonstrates that *InDS* outperforms four cutting-edge works in terms of acceptance ratio and revenue-to-cost ratio, along with controlling network congestion effectively, thereby improving the overall network performance. This superiority is demonstrated by a 28% improvement in acceptance and 45% in revenue-to-cost ratio. This highlights *InDS* assurance as a resilient and efficient solution to the VNE problem in the context of NV.

Future directions for the *InDS* involves integrating enhanced security and QoS considerations, and to ensure improved performance, we intend to switch to real-time data testing. Also, we are planning to include additional network features during the training to refine *InDS* robustness and versatility. Additionally, we also aim to include testing and validating *InDS* on commercial cloud platforms such as AWS and Azure.

REFERENCES

[1] M. Minardi, T. X. Vu, I. Maity, C. Politis, and S. Chatzinotas, "Traffic-aware virtual network embedding with joint load balancing and datarate assignment for SDN-based networks," *IEEE Trans. Netw. Service Manage.*, early access, Jan. 11, 2024, doi: 10.1109/TNSM.2024.3353079.

[2] A. Satpathy, M. N. Sahoo, L. Behera, and C. Swain, "ReMatch: An efficient virtual data center re-matching strategy based on matching theory," *IEEE Trans. Services Comput.*, vol. 16, no. 2, pp. 1373–1386, Mar. 2023.

[3] T. G. K. Kumar, H. K. Virupakshaiah, and K. Nanda, "Ensuring an online chat mechanism with accountability to sharing the non-downloadable file from the cloud," in *Proc. 2nd Int. Conf. Appl. Theor. Comput. Commun. Technol. (iCATccT)*, Jul. 2016, pp. 718–721.

[4] P. Zhang, C. Wang, C. Jiang, and A. Benslimane, "Security-aware virtual network embedding algorithm based on reinforcement learning," *IEEE Trans. Netw. Sci. Eng.*, vol. 8, no. 2, pp. 1095–1105, Apr. 2021.

[5] T. G. K. Kumar, S. K. Addya, A. Satpathy, and S. G. Koolagudi, "NORD: NOde ranking-based efficient virtual network embedding over single domain substrate networks," *Comput. Netw.*, vol. 225, Apr. 2023, Art. no. 109661.

[6] T. G. K. Kumar, A. Srivastava, A. Satpathy, S. K. Addya, and S. G. Koolagudi, "MatchVNE: A stable virtual network embedding strategy based on matching theory," in *Proc. 15th Int. Conf. Commun. Syst. Netw. (COMSNETS)*, Jan. 2023, pp. 355–359.

[7] P. Satish, D. Lingraj, S. Anjan Kumar, and T. G. Keerthan Kumar, "Comparison of D-vine and R-vine techniques for virtual network embedding problem," *IOP Conf. Ser., Mater. Sci. Eng.*, vol. 1187, no. 1, Sep. 2021, Art. no. 012035.

[8] T. G. K. Kumar, S. Tomar, S. K. Addya, A. Satpathy, and S. G. Koolagudi, "EFraS: Emulated framework to develop and analyze dynamic virtual network embedding strategies over SDN infrastructure," *Simul. Model. Pract. Theory*, vol. 134, Jul. 2024, Art. no. 102952.

[9] P. Zhang, "Incorporating energy and load balance into virtual network embedding process," *Comput. Commun.*, vol. 129, pp. 80–88, Sep. 2018.

[10] H. Cao, L. Yang, and H. Zhu, "Novel node-ranking approach and multiple topology attributes-based embedding algorithm for single-domain virtual network embedding," *IEEE Internet Things J.*, vol. 5, no. 1, pp. 108–120, Feb. 2018.

[11] P. Zhang, H. Yao, and Y. Liu, "Virtual network embedding based on computing, network, and storage resource constraints," *IEEE Internet Things J.*, vol. 5, no. 5, pp. 3298–3304, Oct. 2018.

[12] M. Pham, D. B. Hoang, and Z. Chaczko, "Congestion-aware and energy-aware virtual network embedding," *IEEE/ACM Trans. Netw.*, vol. 28, no. 1, pp. 210–223, Feb. 2020.

[13] F. Yan, T. T. Lee, and W. Hu, "Congestion-aware embedding of heterogeneous bandwidth virtual data centers with hose model abstraction," *IEEE/ACM Trans. Netw.*, vol. 25, no. 2, pp. 806–819, Apr. 2017.

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

[15] P. Zhang, C. Wang, N. Kumar, W. Zhang, and L. Liu, "Dynamic virtual network embedding algorithm based on graph convolution neural network and reinforcement learning," *IEEE Internet Things J.*, vol. 9, no. 12, pp. 9389–9398, Jun. 2022.

[16] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 29, 2016, pp. 3837–3845.

[17] S. K. Maurya, X. Liu, and T. Murata, "Simplifying approach to node classification in graph neural networks," *J. Comput. Sci.*, vol. 62, Jul. 2022, Art. no. 101695.

[18] R. Zhou and E. A. Hansen, "Breadth-first heuristic search," *Artif. Intell.*, vol. 170, nos. 4–5, pp. 385–408, Apr. 2006.

[19] T. G. K. Kumar, S. K. Addya, and S. G. Koolagudi. (Feb. 2024). *InDs*. [Online]. Available: https://github.com/Keerthankumar22/InDs.git

[20] K. T. Nguyen, Q. Lu, and C. Huang, "Rethinking virtual link mapping in network virtualization," in *Proc. IEEE 92nd Veh. Technol. Conf. (VTC-Fall)*, Nov. 2020, pp. 1–5.

[21] Q. Hu, Y. Wang, and X. Cao, "Virtual network embedding: An optimal decomposition approach," in *Proc. 23rd Int. Conf. Comput. Commun. Netw. (ICCCN)*, Aug. 2014, pp. 1–6.

[22] S. Shanbhag, A. R. Kandoor, C. Wang, R. Mettu, and T. Wolf, "VHub: Single-stage virtual network mapping through hub location," *Comput. Netw.*, vol. 77, pp. 169–180, Feb. 2015.

[23] H. Yao, X. Chen, M. Li, P. Zhang, and L. Wang, "A novel reinforcement learning algorithm for virtual network embedding," *Neurocomputing*, vol. 284, pp. 1–9, Apr. 2018.

[24] M. Pham, D. B. Hoang, and Z. Chaczko, "Realization of congestion-aware energy-aware virtual link embedding," in *Proc. 29th Int. Telecommun. Netw. Appl. Conf. (ITNAC)*, Nov. 2019, pp. 1–6.

[25] M. Dolati, S. B. Hassanpour, M. Ghaderi, and A. Khonsari, "DeepViNE: Virtual network embedding with deep reinforcement learning," in *Proc. IEEE INFOCOM Conf. Comput. Commun. Workshops (INFOCOM WKSHPS)*, Apr. 2019, pp. 879–885.

[26] Z. Duan and T. Wang, "Towards learning-based energy-efficient online coordinated virtual network embedding framework," *Comput. Netw.*, vol. 239, Feb. 2024, Art. no. 110139.

[27] T. G. K. Kumar, K. Aneesh, A. Siddheshwar, A. Marali, A. Kamath, S. G. Koolagudi, and S. K. Addya, "DeepVNE: Deep reinforcement and graph convolution fusion for virtual network embedding," in *Proc. 16th Int. Conf. Commun. Syst. Netw. (COMSNETS)*, Jan. 2024, pp. 633–636.
[28] H. Xu and B. Li, "Anchor: A versatile and efficient framework for resource management in the cloud," *IEEE Trans. Parallel Distrib. Syst.*, vol. 24, no. 6, pp. 1066–1076, Jun. 2013.
[29] T. Opsahl, F. Agneessens, and J. Skvoretz, "Node centrality in weighted networks: Generalizing degree and shortest paths," *Social Netw.*, vol. 32, no. 3, pp. 245–251, Jul. 2010.
[30] C. L. Moreira, C. A. Kamienski, and R. A. C. Bianchi, "5G and edge: A reinforcement learning approach for virtual network embedding with cost optimization and improved acceptance rate," *Comput. Netw.*, vol. 247, Jun. 2024, Art. no. 110434.
[31] D. Medhi and K. Ramasamy, *Network Routing: Algorithms, Protocols, Architectures*. San Mateo, CA, USA: Morgan Kaufmann, 2017.
[32] R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 12, 2000, pp. 1057–1063.
[33] W. F. Donoghue, *Distributions and Fourier Transforms*. New York, NY, USA: Academic, 2014.
[34] S. Mehra, G. Raut, R. D. Purkayastha, S. K. Vishvakarma, and A. Biasizzo, "An empirical evaluation of enhanced performance softmax function in deep learning," *IEEE Access*, vol. 11, pp. 34912–34924, 2023.
[35] H. Shen, K. Zhang, M. Hong, and T. Chen, "Towards understanding asynchronous advantage actor-critic: Convergence and linear speedup," *IEEE Trans. Signal Process.*, vol. 71, pp. 1–11, 2023.
[36] P. Zhang, Z. Luo, N. Kumar, M. Guizani, H. Zhang, and J. Wang, "CE-VNE: Constraint escalation virtual network embedding algorithm assisted by graph convolutional networks," *J. Netw. Comput. Appl.*, vol. 221, Jan. 2024, Art. no. 103736.
[37] H. Yao, S. Ma, J. Wang, P. Zhang, C. Jiang, and S. Guo, "A continuous-decision virtual network embedding scheme relying on reinforcement learning," *IEEE Trans. Netw. Service Manage.*, vol. 17, no. 2, pp. 864–875, Jun. 2020.
[38] S. Wang, J. Bi, J. Wu, A. V. Vasilakos, and Q. Fan, "VNE-TD: A virtual network embedding algorithm based on temporal-difference learning," *Comput. Netw.*, vol. 161, pp. 251–263, Oct. 2019.
[39] M. Rost and S. Schmid, "Virtual network embedding approximations: Leveraging randomized rounding," *IEEE/ACM Trans. Netw.*, vol. 27, no. 5, pp. 2071–2084, Oct. 2019.
[40] R. Matthias, E. Alexander, and D. Elias. (2020). *Alib*. [Online]. Available: https://github.com/vnep-approx/alib
[41] S. Knight, H. X. Nguyen, N. Falkner, R. Bowden, and M. Roughan, "The Internet topology zoo," *IEEE J. Sel. Areas Commun.*, vol. 29, no. 9, pp. 1765–1775, Oct. 2011.
[42] S. Knight, N. Falkner, H. X. Nguyen, P. Tune, and M. Roughan, "I can see for miles: Re-visualizing the Internet," *IEEE Netw.*, vol. 26, no. 6, pp. 26–32, Nov. 2012.

T. G. KEERTHAN KUMAR (Graduate Student Member, IEEE) received the B.Tech. and M.Tech. degrees in CSE from Visvesvaraya Technological University, Karnataka, India. He was with the Cloud and Smart System Services Laboratory, Department of CSE, National Institute of Technology Karnataka, Surathkal, India. He was with Dell Research and Development, Bangalore, and VMware Software India Private Ltd., Bangalore. He is currently an Assistant Professor with the Department of ISE, Siddaganga Institute of Technology, Karnataka. His current research interests include cloud computing, the Internet of Things, distributed systems, and artificial intelligence.

SOURAV KANTI ADDYA (Senior Member, IEEE) received the Ph.D. degree in CSE from NIT Rourkela, India. He was a Postdoctoral Fellow with the Department of CSE, IIT Kharagpur, India. He is currently an Assistant Professor of CSE with NITK Surathkal, India, where he is heading the Cloud and Smart System Services Laboratory. His technical interests include cloud systems, the IoT, and blockchain. He is a member of ACM.

SHASHIDHAR G. KOOLAGUDI (Member, IEEE) received the Ph.D. degree from IIT Kharagpur. He has worked on using different machine learning algorithms for processing big data applications, such as speech processing during the Ph.D. degree. He is currently an Associate Professor with NITK Surathkal. He has around 20 years of experience in the fields of teaching and research. His research interests include machine learning, signal processing, and big data analytics.