NetAI-Gym: Customized Environment for Network to Evaluate Agent Algorithm using Reinforcement Learning in Open-AI Gym Platform

Varshini Vidyadhar[1]
Research Scholar, Department of Computer Science and Engineering
Bangalore Institute of Technology
Bangalore, India

Dr. Nagaraj R[2]
Professor, Department of Information Science and Engineering, Bangalore Institute of Technology, Bangalore, India

Dr. D V Ashoka[3]
Professor, Department of Information Science and Engineering, JSS Academy Technical Eduction,Bangalore, India

***Abstract*—The growing size of the network imposes computational overhead during network route establishment using conventional approaches of the routing protocol. The alternate approach in contrast to the route table updating mechanism is the rule-based method, but this also provides a limited scope in the dynamic networks. Therefore, reinforcement learning promises a better way of finding the route, but it requires an evaluation platform to build a model synchronization between route and agent. Unfortunately, the de-facto platform for agent evaluation, namely Open-AI Gym, does not provide a suitable networking environment. Therefore, this paper aims to propose a networking environment as a novel contribution by designing a suitable customized environment for a network synchronically with Open-AI Gym. The successful deployment of the proposed network environment: NetAI-Gym provides a functional and practical result that can be used further to develop routing mechanisms based on Q-learning. The validation of the proposed NetAI-Gym is carried out with different nodes in the network regarding Episodes Vs. Reward. The experimental outcome justifies the validity of the proposed NetAI-Gym that it is suitable for solving network-related problems.**

***Keywords*—Open-AI Gym; network; environment; agent; reinforcement learning**

I. Introduction

Artificial intelligence (AI) is being explored way back in 1997 for some problems like exploring the possibility of adaptive-AI using a network of neurons like adaptive elements, where the focus of the study was on the adaptive systems, where the learning system adapts some behavior from the environment to maximize the signal. It is being observed that this approach has received very little attention from the researchers from the computational perspectives [1][2]. At the same time, the same idea of the hedonistic-learning system (HLS) of that time has been realized today as Reinforcement Learning (RL). However, with a hypothesis that data are collected only from the IEEE Digital library. It is found that the routing problem in the network became an active research problem in the last 20 years, with an overall publication of 86,226. It is observed that in the last decade, the total publication for the same problem is 52,344, which is alone 60.7%, which shows that the active focus of the researchers is higher in the running decade. Considering this 60.7% data as 100% and then the stake of Reinforcement Learning is found only 322 in totality, which is hardly 0.6% and 0.3% from last two decade. Therefore, it can be concluded that more efforts are required to study and develop a solution paradigm for routing problems in a network using reinforcement learning. The typical architecture of reinforcement learning is shown in Fig. 1.

The basic design of RL includes building two functional blocks, namely E and Ag. The Ag takes appropriate Ac based on the O provided by E and subsequently based on the Ac taken, E gives positive or negative R. Therefore, to evaluate the agent algorithm, a suitable platform of the environment is required as per the domain context and particular task. The role of RL is to solve the problem of sequential decision tasks in different networking scenarios. There are many methods found in the literature to solve this problem by using i) Game theory [3], ii) Swarm [4], iii) a probabilistic technique [5], and many more [6-7]. However, all these approaches are associated with some advantages and limitations. But RL can be utilized to address very complex problems that conventional approaches cannot address. RL refers to the computer intelligence field that studies programmed computing procedures and dynamically optimizes their performance based on experience learned from the environment. Therefore, RL offers promising context that can be used to develop adaptive mechanisms for network routing so that better performance can be achieved on complex problems without performing any engineering particular to the problem. RL's logic considers a decision-maker component (agent) in the environment (set of states with inputs). At every step, the agent takes action and gets observations and rewards when interacting with the environment. The RL algorithm tries to maximize a certain amount of reward achieved by the agent. The RL environment for networking was configured based on a general backbone network according to the concept of a partially observable Markov decision process [8]. However, most of the researchers failed to produce their experiments based on the RL. Recently, an introduced RL tool kit, namely Open-AI Gym, removes this problem and lacking standardization in the research process by giving versatile numbers of the environment with great ease of setting up. This toolkit offers a collection of test problems. It concentrates on RL's scenario setting, in which the experience learned by the agent is divided into several episodes. The Open-AI provides a benchmarking framework for building and testing RL algorithms. However, to date, no any Open-AI Gym library is available for networking. A thorough investigation of the existing Open-AI Gym platform reveals that it has 7 explicit classes contributed by Open-AIGym and an additional class of third-party contribution. A detailed explanation is given in section III. It is found that the available environment in Open-AI Gym is not applicable for solving the problem of network, especially the routing. Researchers use various network simulators and experimental testbeds. Hence, this paper is the first of its type to contribute a custom design of an environment for evaluating network routing using RL on the Open-AI Gym platform. The proposed NetAI-Gym offers a scalable networking environment for implementing any reinforcement learning algorithm for training agents and accessing their performances in the context of networking. This paper is organized as in Fig. 2.

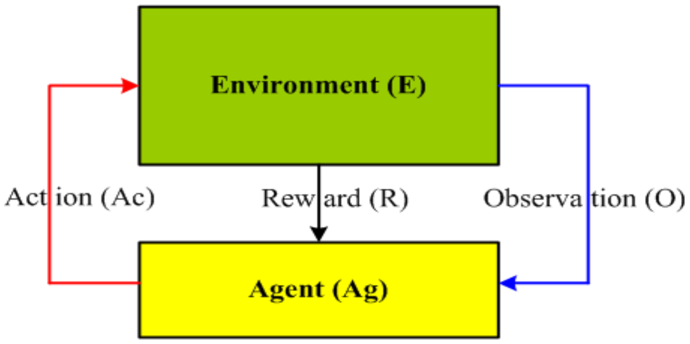


Fig. 1. Typical Reinforcement Learning Architecture.

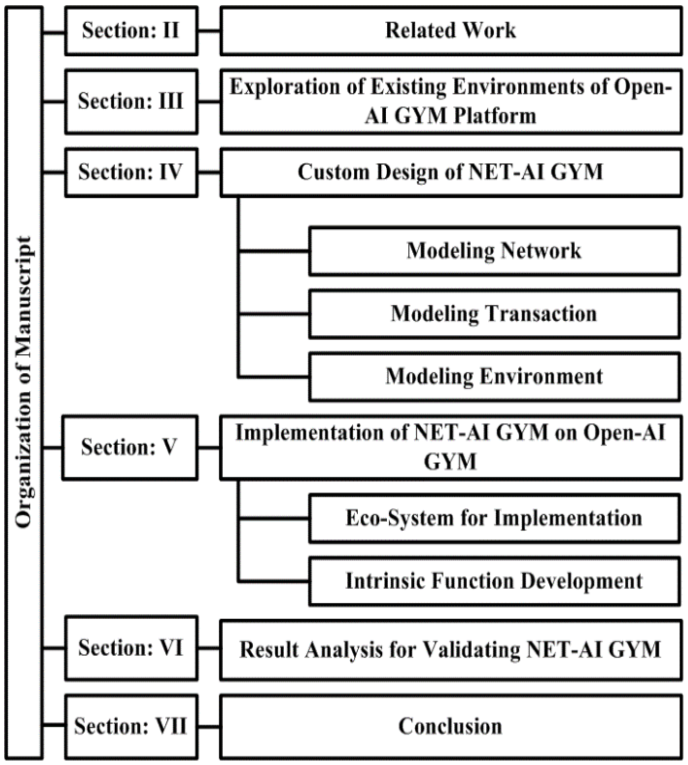

Organization of Manuscript
Section: II
Related Work
Section: III
Exploration of Existing Environments of Open-AI GYM Platform
Section: IV
Custom Design of NET-AI GYM
Modeling Network
Modeling Transaction
Modeling Environment
Section: V
Implementation of NET-AI GYM on Open-AI GYM
Eco-System for Implementation
Intrinsic Function Development
Section: VI
Result Analysis for Validating NET-AI GYM
Section: VII
Conclusion

Fig. 2. Organization of the Paper.

II. RELATED WORK

The use of RL is found in the literature for the various task in the respective networks. In WSN, automation of the radio scheduling task for optimizing energy usage is designed using the RL technique [9]. RL eliminates overheads of the control messages used for communication among neighbors as ∀ nodes ∈WSN as it approximates its neighbor conditions based on the current state. Since this method does not use any specific tool like Open-AI Gym to evaluate the agent designed, it uses an approach of trial and error to simplify the problem of the sequential decision using game theory. However, it is a computationally expensive method. Reinforcement learning aims to solve sequential decision tasks through trial-and-error interactions with the environment. Another related network in the context of industrial-WSN uses RL for minimizing the latency and maximizing the lifetime of the network [10]. There remain many open issues that include computation of time complexities in terms of learning and exploration that validate performance enhancements. A Markov decision process is used to formulate a path selection process. A deep RL technique is then applied to minimize the probability of network congestion in the network under heavy traffic [11]. Throughput is achieved, but overall network performance can be enhanced by considering network-related dynamic parameters. RL technique adoption is also found in [12] for designing grid-oriented routing mechanisms to address message forwarding challenges in the Vehicular ad-hoc network (VANET). However, the focus is only on the problem of the message forwarding issue from the source to the fixed destination. The Q-table is learned offline and may not be well suited to the dynamic characteristics of urban VANETs. Traffic-aware and road-side-unit (RSU) supported routing mechanism is introduced in [13]. Here, RL is implemented to facilitates intelligent data transmission processes between vehicle-to-vehicle and RSU-2-RSU.However, the vehicle's direction is not considered, which may affect the performance of the routing scheme when it comes to real-time deployment scenarios. An RL-based routing protocol is designed in [14] to analyze the impact of a varying number of nodes on the performance of the Underwater Acoustic Sensor Networks (UWSN). Q-learning is used, where the node has packets to forward based on the state of the buffer, remaining energy, and proximity of adjacent nodes to select the next sender node. In [15], a channel-aware RL-oriented adaptive path selection technique is introduced for multi-hop UWSN. The protocol switches between single-path and multi-path routing accomplish joint optimization in energy consumption and packet delivery ratio. Q-learning-based distributed opportunistic routing mechanism is introduced by [16] for minimizing the average packet routing cost in Wireless Ad-hoc network. This mechanism combinedly solves the problems related to learning and routing network structure is characterized by the communication and data transaction success probability. An efficient mechanism for collaborative RL is used in [17] for optimizing path selection in MANET. The use of RL is used for routing optimization in software-defined networking (SDN) [18]. The effectiveness of the agent is tested under the self-convergence aspects. The authors in [19] used a deep RL mechanism for optimizing routing performance in SDN. The authors in [20] explored RL's effectiveness towards the energy harvesting routing model based on Q-learning for multi-hop Cognitive Radio networks (CRN). The runtime complexity of this model is $O(N^2)$. However, the performance of CRN can be further enhanced using spectrum sensing and power allocation mechanism. The problem of link selection in the Energy Harvesting Relay network (EHRN) is solved by [21] using RL and Deep-Q-learning techniques. A pre-trained algorithm is used to avoid the massive iterations and alleviate the computation overhead in convergence optimization. However, this approach is not much scalable when environmental parameters change. The use of RL in [22] is found for designing routing protocol in Magnetic Induction Underwater Sensor Networks (MIUSN). A Q-table is derived by taking into account the distance factor and energy loss. However, the protocol requires periodic control message exchange for neighbor discovery to give rise to high overheads and reduce channel usage due to slow propagation. The adoption of the RL-based Q-learning technique for content placement is found in [23] for a dynamic cloud content delivery network (CCDN). An efficient routing design based on RL in Unmanned Robotic Network (URN) is suggested by [24] considering location information, link condition, and battery information to realized the neighbor node with the most significant future reward for determining the next hop. Table I summarizes the above-discussed literature for quick insight concerning network scenarios and issues handled.

It is analyzed that RL is mainly adopted for addressing optimization problems of routing and congestion control in various networking scenarios. However, none of the studies are found in the context of using a customizable environment for the agent algorithm designed to solve routing and network performance problems using RL. The environment used in the existing literature is formulated based on the general core network in the simulation process. Since RL is associated with reasonable overhead, the existing approaches are not suitable for providing an efficient solution. It may be exposed to many problems when it comes to real-time deployment scenarios. Therefore, a customizable environment specific to RL approaches for networking is required for suitable analysis of performance.

TABLE I. SUMMARY OF EXISTING LITERATURE

Authors	**Network Scenario**	**Problem Handled**
5, 6	WSN	Energy usage [5], Latency [6]
7	Relay	Network congestion
8,9	VANET	Message forwarding [8], transmission efficiency [9]
10,11	UWSN	Energy
12	Ad-Hoc	Routing cost
13	MANET	Path selection
14,15	SDN	Routing Performance
16	CRN	Network Performance
17	EHRN	Computation overhead
18	**MIUSN**	Energy and Channel utilization
19	CCDN	Content placement
20	URN	Routing and Energy

III. Exploration of Existing Environments of Open-AI Gym Platform

The first instance of Open-AI Gym is found in 2016 to support reinforcement learning used for various decision making and control systems. It includes a study of agent learning to achieve the learning goal in an uncertain and complex environment. The use of RL is found in the diversified problem domain, such as robot motor control, games, and business decision makings, wherever a sequence of decision making is required. In the recent past, RL is being used in various complex environments. However, the advancement into deep learning demands engineering aspects specific to the problem.

A. Custom Design of Net-Ai Gym

1) Modelling network: The mathematical model for the network is represented by a collection of vectors as in set:$\eta = \{\overrightarrow{N_1}, \overrightarrow{N_2}, \overrightarrow{N_3}, \cdots \overrightarrow{N_n}\}$, where, $\forall \overrightarrow{N_k} \in \eta$ represents a node with two intrinsic properties as {Node number (X_k), Set of links(R_k)} s.t $\overrightarrow{N_k}$ is represented by a pair of $\{X_k, R_k\}$, where k =1to n, $n \in \mathbb{N}$ and $n \geq 2$. Basically, a node $\overrightarrow{N_k}$ may have connectivity with many of the node $\in \eta - \{\overrightarrow{N_k}\}$, therefore, $\forall$ the link-set is represented by a collection of vectors: $R_l = \{\overrightarrow{L_1}, \overrightarrow{L_2}, \overrightarrow{L_3}, \cdots \overrightarrow{L_m}\}$ s.t $\forall \overrightarrow{L_k} \in R_l$ contains two intrinsic properties as {Connecting node number (X_k), the weight of the link (W_k)}. This arrangement of vector representation for the nodes and reference to the links alleviates the challenge of handling memory usage by dimensionality reduction. Else the simple representation of Tensor imposes excessive use of memory and computational resources. Fig. 3 illustrates a sample representation of a weighted 3 node network with 3 links.

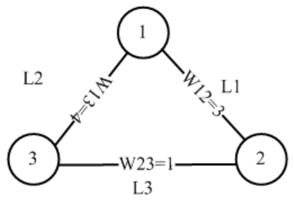


Fig. 3. Network Representation.

$\eta = \{\{X1, R1\}, \{X1, R2\}, \{X1, R3\}\}$, where, R1 → {{X2,3}, {X3,4}}, R2→{{X1,3}, {X3,1}}, R3→ {{X1,4}, {X2, 1}}

2) Modeling transaction: The transaction (T) is the delivery of one packet (P) from the source node (Ns) to the destination node (Nd). There are two outcomes of the transaction: i) FAILURE and ii) SUCCESS. To elaborate the process of the transaction and associate states, Fig. 4 with 4-nodes considering node N1 as Ns and node N4 as Nd.

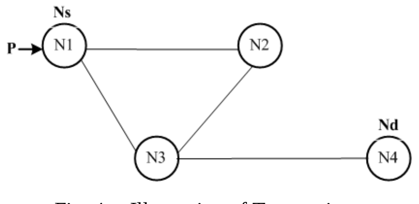


Fig. 4. Illustration of Transaction.

There are various possibilities of transactions between N1 to N4. The first possible condition is that when node N1 does not find node N4 in its first transaction, then it DROPS the packet P, and the state is flagged to FAILURE. Based on the previous record of T, in the next transaction, N1 will look for another node, say node N2, and this time also the state is FAILURE as when a packet is delivered to N2, no connection is found between node N2 and N4. Therefore, in reference to Fig. 3, the only possible transaction for SUCCESS state is to transfer packet P either from N1 (Ns) → N2→ N3→N4 (Nd) or from N1 (Ns) → N3→N4 (Nd) and in this way packet is DELIVERED from Ns to Nd. The computation process for one transfer where a transfer is a process of forwarding a packet from one node to another node is as in algorithm 1.

Algorithm-1: Computational process of transfer **T(Nk, Nk+1)**

```
Input: η
Output: REWARD, DONE
Start
1. Initialize Nk, Nk+1
2. T (Nk→Nk+1 )
3. GET Rk∈ $Nk$
4. GET Xk+1 ∈ Rk
5.    If Xk+1 ∈ Rk
          If Xk+1 is $X_D$
          REWARD = V$l$∈N
            DONE = True
          Else
              REWARD = -Wk
              DONE = False
      Else
          REWARD = 0
          DONE = True
6. Return: REWARD, DONE
End
```

In the designed networkη, the computational process for one transfer is the conditions, the environment to get respective REWARD () and DONE () subjected to network conditions of node connectivity. The process computes the transaction between node Nk→ Nk+1 as T (Nk→Nk+1). Typically, REWARD is the value returned by the algorithm that signifies encouragement if it is positive REWARD and discouragement if it is negative REWARD towards selecting the same route in the next transaction. In contrast, DONE is a final state which signifies either FAILURE or SUCCESS before restarting the next transaction. The respective values of Rk from the Nk set and Xk+1 from Rk is obtained.

Further, firstly it checks that Xk+1 is an element of Rk. If this condition is found to be true then, it checks whether Xk+1 is X_D (destination node number) or not. If it is true, then a large natural number is assigned as REWARD, and the transaction state DONE is set to True means it goes to the next iteration. Otherwise, a negative value is assigned to REWARD, and the transaction state DONE is set to False, which means the transaction further continues. In case Xk+1 does not belong to Rk, then zero is assigned as REWARD, and Transaction state DONE is set to True for the further transfer iterations of the transaction.

3) Modeling environment: The typical environment modeling for Net-AI Gym mimics Fig. 1, and corresponding constructs are mapped as {E, Ag}: → {O, R, Ac} and in the case of network, the action (Ac) is mapped to movement of the packets from the node (Ni) to node (Nj) as shown in Fig. 5.

Typically, there are many state or observations between starting observation to the terminating observation, and the set of ∀ Ob ∈ {Ob-start, Ob-next . . . Ob-Terminating} is known as one episode. In the custom design of NetAI-Gym, one episode (Eps) is a journey of a packet (P) from the source node (Ns) to the destination node (Nd), and the Eps ends when either the packet drops or it reaches the Nd.The underlying architecture of the environment adopts the Markov decision process and works in a stochastic manner following the finite state machine, as shown in Fig. 6.

In Fig. 5, the states set {S1, S2, S3 …Sn … Sd} is mapped with the respective nodes set {N1, N2, N3 …Nn … Nd}, and another state considered is Done. The possible transitions are: {Transfer, Drop, Reset, Delivered}. The network behavior exhibits randomness as the weight of the links varies due to various noises, whereas the route establishment process is entirely in control of the agent. Therefore, the Markov decision process (MDP) is justified as MDP is suitable for scenarios where the possible outcomes are influenced jointly by random variables and decision-makers. The purpose of this particular MDP is also to find the best policy (π) for the decision-maker, such that Nk = $\pi(N_{k-1})$ where the transfer gives the maximum reward.

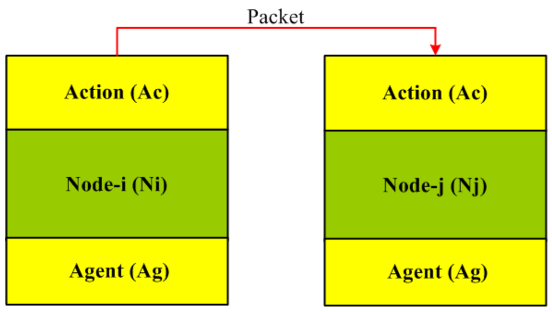


Fig. 5. Movement of Packets: Action from Ni to node Nj.

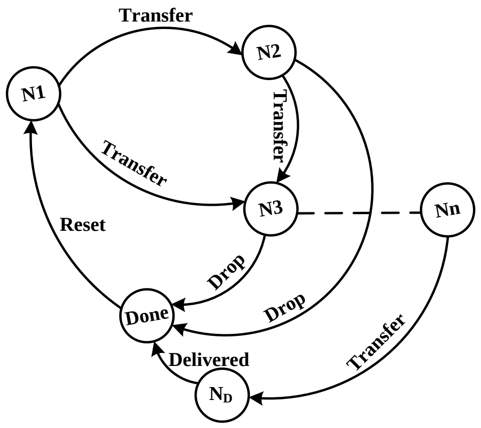


Fig. 6. Finite State Machine for Environment-Net-AI Gym.

However, since randomness is involved, a maximum REWARD cannot be ensured for every transfer. Hence, the model is designed to REWARD a huge positive value when the destination state is reached. As shown in Fig. 5, the packet may drop from any node. Hence a transition is defined from all states to DONE state as a DROP. This transition represents that any node may DROP the packet. The delivered transition is defined only from the destination state to the DONE state. This transition represents that the transaction is successful only when the packet reaches the destination node. Once it is DONE, the network must handle the next packet. The RESET transition from the DONE state shows this to the first state (Ns).

Generally, the best policy π is a function or a transformation that outputs the next state when the present state is given. This is done by either a lookup table or a function approximator to save space. The agent has to find π. This environment is being modeled to help the agent to find π efficiently. For that to happen, the model must be designed efficiently so that maximum reward is awarded when the agent reaches the destination, and it should discourage alternative non-efficient policies. Hence, the model is designed to award negative rewards when the agent tries to follow the longer path. The agent keeps exploring until no better policy exists compared to the present policy. And due to this, a well-programmed agent always finds the best policy. The environment is written so that there will always be a better policy with higher reward as long as the agent finds the best policy. The environment also ends when the agent makes a mistake. Thereby allowing the agent to learn how to ensure the packet is not dropped. All these are written keeping in mind both ML-based as well as rule-based agents. The environment is modeled so that performance doesn't deteriorate even if we scale the model. The model is made highly scalable since the overall state machine architecture is quite simple. The randomness is also modeled to simulate real-world scenarios of network disturbances. Overall, this model simulates a real-world computer network as realistically as possible.

IV. Implementation of Net-AI Gym on Open-AI Gym

This section presents a detailed discussion on the modeling and implementation design of the proposed RL environment for networking, namely 'Net-AI GYM'. The discussion first highlights the computational ecosystem required an intrinsic function for Net-AI Gym development followed by algorithmic steps and discussion.

A. *Ecosystem for the Implementation*

A professional virtual environment management tool, Anaconda is used to build the custom version of the Open-AI Gym library as Net-AI Gym. Anaconda helps to organize the required libraries, including i well) core Python-3.8 for scripting, ii) Pandas to acquire and handle data, iii) NumPy for handling complex matrix manipulations, iv) Matplot lib for visualization through plotting. Apart from these packages, the essential package used is a NetworkX for building and managing network representation using graph theory. Since the Net-AI Gym is aimed to be used for solving various network-related problems using reinforcement learning. Therefore, to evaluate the performance of the designed agents, the stack of computational ecosystem preferred is as in Fig. 7.

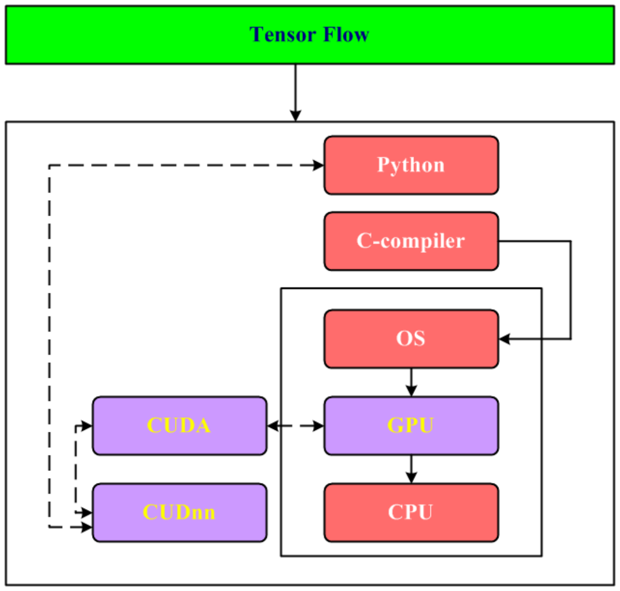


Fig. 7. Stack of Computational System for ML.

The above stack requires intrinsic operational support for complex and large matrix manipulations. Therefore, to speedup the training process, Nvidia-GTX architecture supported GPU issued along with a CPU with the Intel(R) Core(TM) i7-9750H CPU @ 2.60GHz 2.59 GHz with 16 GB of RAM, a best-suited trailing architecture to the next best possible in the market. TensorFlow is the neural network library by Google, used to create agents to work with the environment. This can even be used to benchmark the environment; hence we need it at the top layer. As shown in Fig. 7, there are some optional components, and there are some compulsory components. A GPU is used to handle complicated matrix operations and makes the program run faster as most of the ML operations are mainly Matrix multiplication. However, even the CPU can handle matrix operations, and GPU is not a must. However, if A GPU is being used, TensorFlow must access it. CUDA is a c library given by Nvidia, allowing the C compiler access to GPU. To allow python to access GPU, we must install the CUDnn package, which acts as a wrapper to CUDA for python. If GPU access is allowed in python, CUDA will automatically access GPU.

The implementation process can proceed once the above environment is set up. To implement this environment, the NetworkX library in python is mandatory. This is required since the proposed study uses approach graph theory in this implementation. NetworkX provides an excellent source of graph theory implementation and calculations. The Graph (G) in the NetworkX contains many nodes, and each node can be treated as any bashable object. In our case, A node is nothing but an integer representing the node number. Each connection can have weight. Even though bidirectional weight is allowed in NetworkX, this feature is not being used in this implementation. During implementation, the weight from Node A to B and vice versa should be the same. This operation is ensured programmatically. NetworkX is the best-suited library to implement this environment, and more information can be stored in each node and edges by using various bashable objects in future work.

B. Intrinsic Function Development

The customization of the proposed Net-AI Gym environment includes the typical process as in Fig. 8. The flow of the environment is as shown in figure x. The init function is executed only once since the environment is initiated only once. The reset function is executed at the end of every episode. Each episode represents a transaction, i.e., the movement of a packet from the source node to the destination node. The step function is executed in a loop till the transaction is over. The step function represents a transfer, i.e., the movement of a packet from one node to another. The render function is optional as it is used only to visualize the network and transaction. Only when the learning needs to be monitored, the render function is activated.

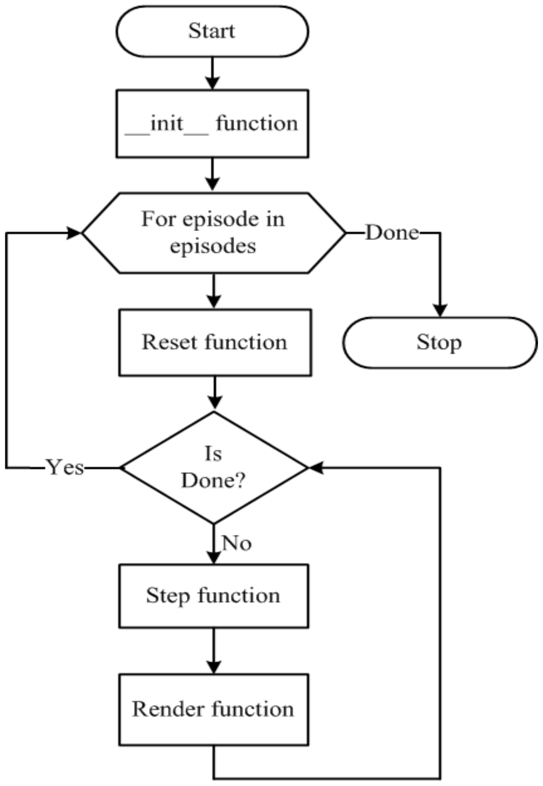


Fig. 8. Flow of Environment for Net-AIGym.

4) Init function: Init function is executed at the beginning of the environment to initiate it. The following algorithm represents the working of the init function.

Algorithm-2: Initiation of Environment
Input: η Output: Network N Start 1. Initialize N 2. Length = len(η) 3. N = {N1,N2,N3..Nn} = createNodes(Length) 4. initalWeights = randomWeights() 5. map(N→ η) 6. return N End

As shown in algorithm2, environment initiation happens by copying the nodes from the data structure to the network graph. Every vector in η is mapped to a node in the network graph. This step is done to ease network operations as the network has built-in functions for network operations. As mentioned earlier, the init function is executed only once. Even if the network gets reset, the network structure stays the same, and nodes stay the same. So only those variables are initialized in this function.

*5) RESET function:*The RESET function is executed on completion of every episode. Meaning, once the packet either drops or reaches the destination, the reset function is executed. During the RESET, all those variables are changing during the execution of the environment.

Algorithm-3: Environment RESET
Input: None Output: Network N Start 1. Nc = Ns 2. Reward = 0 3. Weights = initalWeights 4. done = false 5. return N End

All the variables are reset back to the initial state in the RESET function as shown in the algorithm. DONE is set to false as, during the beginning of the environment, it is not done. The weights are initiated again. The network starts with the initial node itself; the current node (Nc) is set to the source node (Ns) as in the beginning. Since the DONE function is set to false in the RESET function, it can be analyzed from the flowchart; both the STEP and RENDER functions are executed at least once.

6) STEP function: The STEP function is where the actual transaction happens, as shown in algorithm 1. After every step next state, the REWARD and DONE are returned to the agent.

Algorithm-4: STEP function
Input: An (Action) Output: Reward,Done,next state Start 1. Nk = Nc 2. Nk+1 = action 3. Reward,Done = T(Nk,Nk+1) (ref ALG1) 4. Next state = Nk+1 5. return Reward,Done,next state End

The REWARD needs to be given after every step, be it positive or negative. The positive REWARD encourages the agent to follow a similar policy, whereas the negative reward discourages the agent.

7) RENDER function: This function is used to visualize the output of the environment. Every step taken can be visualized. However, if the aim is to simplify the output analysis, this function can be disabled to save time.

Algorithm-5: RENDER function

```
Input: Graph (G), Current Node (Nc)
Output: None
Start
1. for every Nk∈ G
2.     if Nc==Nk
3.         Nk.color = red
4.     else
5.         Nk.color = blue
6. plot(G)
End
```

The render function displays the graph on the screen. If the environment is being used in a Jupyter notebook, the plot needs to be cleared manually.

V. Result Analysis for Validating Net-AI Gym

The default Net-AI Gym with five nodes rendering of the environment is shown in Fig. 9.

Similarly, Fig. 10(a) and Fig. 10(b) illustrate the rendering of the Net-AI Gym with 50 and 100 nodes, respectively, to show the flexibility and scalability of the Net-AI Gym.

Fig. 11 shows the benchmarking of the Net-AI Gym environment with the stable-baselines benchmarking tool.

As shown in Fig. 11, the environment performs well, and an agent can find the path in it. Once the implementation is completed, a stable baselines library [25] is used to benchmark this environment. The agent can get maximum reward only when it finds the best path. For simple 6 node environments, the agents find the best path in 15 episodes. For a complex environment with 100 nodes, the agent finds the best path in 500 episodes.

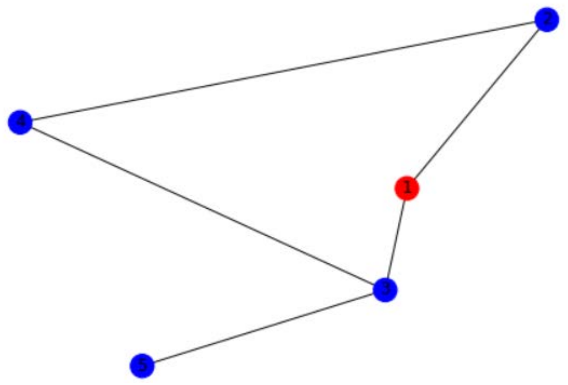
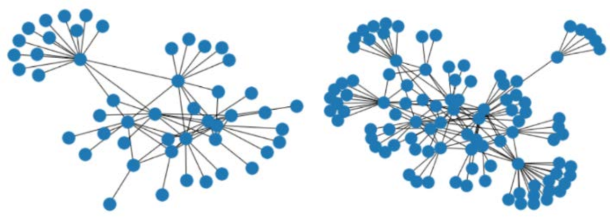


Fig. 9. Rendering of the Environment.



(a): 50 Nodes (b):100Nodes

Fig. 10. (a), (b) Rendering of the Environment with 50 and 100 Nodes.

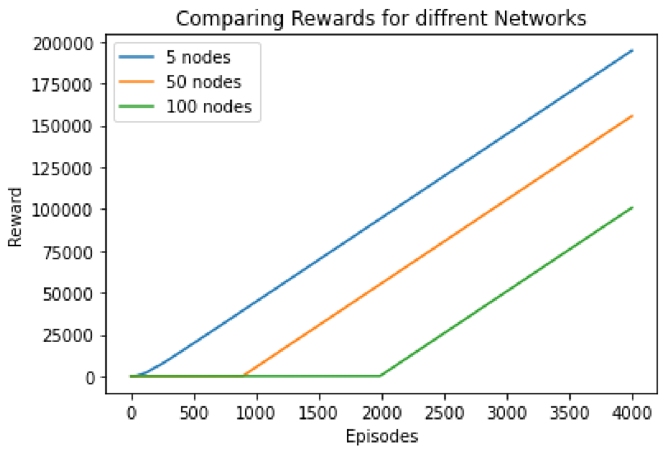


Fig. 11. Baseline Benchmark of Reward Vs. Episodes.

VI. Conclusion

The Open-AI Gym is a de-facto toolbox that provides numerous ready environments to test the agent algorithm's validity. A thorough investigation reveals that the appropriate environment for solving network-related problems is not available to date either by the Open-AI Gym core team or by a third-party contribution. Due to a lack of environment, the potential and advantages of RL-based agents can be fully utilized in networking problems. Therefore, the proposed study has proposed a novel approach of a customized networking environment to support RL-agent to be implemented and tested to solve various networking problems. Our future research problem is to design and develop an optimal routing algorithm for the generic network using reinforcement learning that demands a suitable environment to check the validity of the designed agent. Therefore, the need for an effective and scalable environment, Net-AI Gym, arises. The process of designing Net-AI Gym involves a setting-up stack of computational systems for ML and building a customized function. This function includes __Init__, Reset, Step, Render in the core reposit of Open-AI Gym by adding procedures, such as Transfer, Transaction, Delivered, Dropped, which are as per the requirement of the network routing. The production stage includes registration of environment, re-building Open-AI Gym with registered Net-AI Gym. Finally, the Net-AI Gym environment validation is performed for scalability and proper functioning with default 5 nodes, 50 and 100 nodes. The synchronized support of NetworkX in Net-AI Gym renders the network's visualization successfully and benchmarked with different numbers of nodes 5, 50, and 100 for reward Vs. Episodes analysis shows a stable pattern. Thus, the design and construction of Net-AI Gym provide a suitable platform to evaluate network routing agent algorithms.

References

[1] A.H. Klopf, “Drive-reinforcement learning and hierarchical networks of control systems as models of nervous system function”, *International Journal of Psychophysiology*, Vol. 1(25), pp. 42-3, 1997.

[2] R.S. Sutton, A.G. Barto, “Reinforcement learning: An introduction”, *MIT press*, 2018.

[3] C. Sun, H. Duan, “Markov decision evolutionary game theoretic learning for cooperative sensing of unmanned aerial vehicles”, *Sci. China Technol.* Vol. 58, pp. 1392–1400, 2015.

[4] M. Hüttenrauch, S. Adrian and G. Neumann, "Deep reinforcement learning for swarm systems", *Journal of Machine Learning Research*, Vol. 20(54), pp.1-31, 2018.

[5] R. Ghoul, J. He, S. Djaidja, M. A.A Al-qaness, and S. Kim, "PDTR: Probabilistic and Deterministic Tree-based Routing for Wireless Sensor Networks", *Sensors, 20(6)*, pp.1697, 2020.

[6] J.H. Drake, A. Kheiri, E. Özcan, and E.K. Burke, "Recent advances in selection hyper-heuristics", *European Journal of Operational Research*, Vol. 285(2), pp.405-428, 2020.

[7] Z. A.Aghbari, A.M. Khedr, W. Osamy, I. Arif and D.P. Agrawal, "Routing in wireless sensor networks using optimization techniques: A survey", *Wireless Personal Communications*, pp.1-28, 2019.

[8] H. Wang, N. Liu, Y. Zhang, "Deep reinforcement learning: a survey", *Front Inform Technol Electron Eng*, Vol. 21, pp. 1726–1744, 2020.

[9] J. D. Ye and M. Zhang, "A Self-Adaptive Sleep/Wake-Up Scheduling Approach for Wireless Sensor Networks," in IEEE Transactions on Cybernetics, vol. 48, no. 3, pp. 979-992, March 2018, doi: 10.1109/TCYB.2017.2669996.

[10] G. Künzel, L. S. Indrusiak and C. E. Pereira, "Latency and Lifetime Enhancements in Industrial Wireless Sensor Networks: A Q-Learning Approach for Graph Routing," in IEEE Transactions on Industrial Informatics, vol. 16, no. 8, pp. 5617-5625, Aug. 2020, doi: 10.1109/TII.2019.2941771.

[11] R. Ding, Y. Xu, F. Gao, X. Shen and W. Wu, "Deep Reinforcement Learning for Router Selection in Network With Heavy Traffic," in IEEE Access, vol. 7, pp. 37109-37120, 2019, doi: 10.1109/ACCESS.2019.2904539.

[12] F. Li, X. Song, H. Chen, X. Li and Y. Wang, "Hierarchical Routing for Vehicular Ad Hoc Networks via Reinforcement Learning," in IEEE Transactions on Vehicular Technology, vol. 68, no. 2, pp. 1852-1865, Feb. 2019, doi: 10.1109/TVT.2018.2887282.

[13] J. Wu, M. Fang, H. Li and X. Li, "RSU-Assisted Traffic-Aware Routing Based on Reinforcement Learning for Urban Vanets," in IEEE Access, vol. 8, pp. 5733-5748, 2020, doi: 10.1109/ACCESS.2020.2963850.

[14] Z. Jin, Q. Zhao, and Y. Su, "RCAR: A Reinforcement-Learning-Based Routing Protocol for Congestion-Avoided Underwater Acoustic Sensor Networks," IEEE Sensors Journal, vol. 19, pp. 10881–10891, Nov. 2019.

[15] V. Di Valerio, F. Lo Presti, C. Petrioli, L. Picari, D. Spaccini and S. Basagni, "CARMA: Channel-Aware Reinforcement Learning-Based Multi-Path Adaptive Routing for Underwater Wireless Sensor Networks," in IEEE Journal on Selected Areas in Communications, vol. 37, no. 11, pp. 2634-2647, Nov. 2019, doi: 10.1109/JSAC.2019.2933968.

[16] A. A. Bhorkar, M. Naghshvar, T. Javidi and B. D. Rao, "Adaptive Opportunistic Routing for Wireless Ad Hoc Networks," in IEEE/ACM Transactions on Networking, vol. 20, no. 1, pp. 243-256, Feb. 2012, doi: 10.1109/TNET.2011.2159844.

[17] J. Dowling, E. Curran, R. Cunningham and V. Cahill, "Using feedback in collaborative reinforcement learning to adaptively optimize MANET routing," in IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans, vol. 35, no. 3, pp. 360-372, May 2005, doi: 10.1109/TSMCA.2005.846390.

[18] C. Yu, J. Lan, Z. Guo and Y. Hu, "DROM: Optimizing the Routing in Software-Defined Networks With Deep Reinforcement Learning," in IEEE Access, vol. 6, pp. 64533-64539, 2018, doi: 10.1109/ACCESS.2018.2877686.

[19] Xu, C., Zhuang, W. and Zhang, H., 2020, October. A Deep-reinforcement Learning Approach for SDN Routing Optimization. In Proceedings of the 4th International Conference on Computer Science and Application Engineering (pp. 1-5).

[20] He, X., Jiang, H., Song, Y., He, C. and Xiao, H., 2019. Routing selection with reinforcement learning for energy harvesting multi-hop CRN. IEEE Access, 7, pp.54435-54448.

[21] H. Zhang, D. Zhan, C. J. Zhang, K. Wu, Y. Liu, and S. Luo, "Deep Reinforcement Learning-Based Access Control for Buffer-Aided Relaying Systems With Energy Harvesting," IEEE Access, vol. 8, pp. 145006–145017, Aug. 2020.

[22] Y. Liu, D. Lu, G. Zhang, J. Tian and W. Xu, "Q-Learning Based Content Placement Method for Dynamic Cloud Content Delivery Networks," in IEEE Access, vol. 7, pp. 66384-66394, 2019, doi: 10.1109/ACCESS.2019.2917564.

[23] S. Wang and Y. Shin, "Efficient Routing Protocol Based on Reinforcement Learning for Magnetic Induction Underwater Sensor Networks," in IEEE Access, vol. 7, pp. 82027-82037, 2019, doi: 10.1109/ACCESS.2019.2923425.

[24] W. Jin, R. Gu and Y. Ji, "Reward Function Learning for Q-learning-Based Geographic Routing Protocol," in IEEE Communications Letters, vol. 23, no. 7, pp. 1236-1239, July 2019, doi: 10.1109/LCOMM.2019.2913360.

[25] A. Hill and A. Raffin, M. Ernestus, and A. Gleave, A. Kanervisto, R. Traore, P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, Y. Wu, "Stable Baselines" in GitHub repository on GitHub, 2018, (online: https://github.com/hill-a/stable-baselines