GRouting: Dynamic Routing for LEO Satellite Networks with Graph-based Deep Reinforcement Learning

Hao Wang[1], Yongyi Ran[*2], Lei Zhao[1], Junxia Wang[1], Jiangtao Luo[*2], Tao Zhang[3]
Chongqing University of Posts and Telecommunications, Chongqing, China
[1]{S190101069,S190131061,D170101010}@stu.cqupt.edu.cn, [2]{Ranyy, Luojt}@cqupt.edu.cn,
[3]{zhangtao}@wise-code.com

2021 4th International Conference on Hot Information-Centric Networking (HotICN) | 978-1-6654-0549-2/21/$31.00 ©2021 IEEE | DOI: 10.1109/HOTICN53262.2021.9680855

***Abstract*—The effective and reliable routing for Low Earth Orbit (LEO) satellite networks is intractable. The existing approaches cannot well handle the time-varying topology, frequent link handover, and imbalanced communication load. To tackle these issues, in this paper, we propose *GRouting* algorithm combining Graph Neural Networks (GNN) with Deep Reinforcement Learning (DRL) to dynamically find the optimal routing paths for LEO satellite networks. First, GNN is employed to learn the representation of satellite networks with non-Euclidean data structures. GNN is able to generalize over arbitrary satellite networks topologies, which means that it can deal with time-varying states of satellite networks. Then, based on the representation learned by GNN, DRL is applied to select the optimal routing path between two satellites, which can maximize the utilization of network resources while guaranteeing the requirement of transmission delay. Finally, extensive simulation experiments are carried out to illustrate that 1) our method has a better performance than the baseline algorithms, and 2) the GNN-based method can achieve better generalization over time-varying topologies.**

***Index Terms*—LEO Satellite Networks, Dynamic Routing, Graph Neural Networks, Deep Reinforcement Learning**

I. Introduction

In recent years, Low Earth Orbit (LEO) satellites are attracting more and more attention due to the dramatically increasing requirements of ubiquitous communication. LEO satellites are able to achieve low latency and high throughput communication [1]. A LEO satellite constellation is usually designed for complex communication tasks, which are difficult to be completed by a single satellite. In order to achieve data transmission for a large-scale satellite network, it is critical and fundamental to design an efficient routing scheme.

It is challenging to plan an effective routing scheme for LEO satellite networks due to the following three aspects. First, the topology of satellite networks is time-varying. A LEO satellite constellation usually consists of hundreds of satellites, and each satellite is moving at a high speed, which leads to a constantly changing topology. Thus, routing schemes need to be adapted to changes of the topology. Second, the links between satellites are stochastic and intermittent. Due to the time-varying Inter-Satellite Links (ISLs), channel interference, etc., the communication bandwidth and latency are dynamic, which makes it difficult for traditional static routing strategies to be used in LEO satellite networks. Third, the arrival and demand of the transmission tasks are stochastic. It is difficult to dynamically match network capacity with transmission requirements, which will finally results in workload unbalance and network congestion.

Many efforts have been devoted to addressing the routing problem in LEO satellite networks. Firstly, in order to handle the periodic topology changes, topology virtualization based routing schemes have been developed, including time virtualization [2], [3], space virtualization [4], [5], and space-time virtualization [6]. All of these approaches try to derive a fixed topology and then calculate the routing paths in advance. Although they can shield the movement of satellites and simplify the computation of routes, they can hardly capture the real-time situations in satellite networks and can hardly adjust the routing paths to avoid congestion dynamically [6]. To overcome the shortcomings, dynamic routing schemes have been proposed, which periodically detect and collect network state information on-board to achieve link-aware and/or load-balanced routing [5], [7]–[9]. However, these approaches either take a long time or are unable to get the global state information, thus the calculated routing paths cannot be globally optimal. In addition, these model-based or learning-based dynamic routing approaches usually have poor generalization ability for different topologies and incomplete state information, which will degrade their performance in such a dynamic environment.

To address the issues discussed above, we first adopt the Virtual Node (VN) strategy [4], [5] to construct a logical topology for the LEO satellite network. Then, we propose **GRouting** algorithm combining Graph Neural Networks (GNN) [10] with Deep Reinforcement Learning (DRL) to dynamically find the optimal routing paths for the satellite network. Our main contributions are listed as follows:

- We construct a logical topology via Virtual Node strategy for the target LEO satellite network, and GNN is employed to learn the representation from the logical topology. GNN has a strong generalization ability for different topological structures and edge attributes, which can fully capture the changes of link states.

*The corresponding authors are Yongyi Ran and Jiangtao Luo.
*This work is jointly supported by National Science Foundation of China (No. 62171072, 62172064, 62003067).

- We combine DRL with GNN to dynamically optimize the routing paths for the target LEO satellite network. With the help of GNN, DRL can better perceive the highly dynamic, time-variant and sophisticated environments.
- Extensive experiments based on open datasets are carried out to illustrate that our proposed algorithm can achieve a better performance than the baseline algorithms.

The rest of this paper is organized as follows: Section II summarizes the related work. The system model and problem formulation are presented in Section III. The proposed GRouting algorithm is described in Section IV. Section V shows the evaluation results. Finally, Section VI concludes this paper.

II. Related Work

In this section, we review the related work from the perspectives of topology virtualization and dynamic routing schemes.

A. Topology Virtualization Schemes

Topology virtualization is an efficient way to shield the movement of satellites and simplify the computation of routes. Werner *et al.* [2] presented the concept of Virtual Topology, which makes full use of the periodicity of satellite trajectories to divide the satellite networks into several equal-length time slots, and in each time slot, the topology is deemed as static. Based on Virtual Topology, Jia *et al.* [3] proposed a routing algorithm with Dijkstra and Depth-First-Search algorithms to improve the computation efficiency. Chen *et al.* [4] proposed to deal with the time-varying topology based on Virtual Node, in which physical satellites and the virtual nodes correspond one to one at any time if no satellite failure. In [11], the authors proposed a novel Temporal Netgrid Model (TNM) to portray the time-varying topology of large-scale SSNs. In TNM, the whole space is divided into small cubes (i.e., netgrids) and then, satellites can be located by netgrids instead of coordinates. Based on topology virtualization, the routing paths can be calculated offline. However, topology virtualization schemes can hardly capture the real-time situations in satellite networks and can hardly adjust the routing paths to avoid congestion dynamically.

B. Dynamic Routing Schemes

Many dynamic routing algorithms have been proposed to adapt to the changes of satellite networks and optimize the routing paths dynamically. Liu *et al.* [7] divided the routing schemes into two stages. The preliminary optimal routing paths are first calculated by predicting the satellite networks and then congestion is avoided by informing the upstreaming satellites to reroute a portion of the traffic. Wang *et al.* [8] built a multi-objective optimization model, which adopts modifying factor to adjust path cost and uses congestion prediction to foresee inter-satellite link congestion, and an ant colony algorithm is utilized to solve this model. In [5], the authors proposed a Two-Hops State-Aware Routing Strategy Based on Deep Reinforcement Learning (DRL-THSA) for LEO satellite networks. The Double-Deep Q Network (DDQN) is employed in DRL-THSA to figure out the optimal next hop by inputting the two-hops link states. Sun *et al.* [9] developed an intelligent routing scheme of the SDN satellite networks based on the orthogonal polynomial neural network. These model-based or learning-based dynamic routing approaches usually have poor generalization ability for different topologies and incomplete state information, which will degrade their performance in such a dynamic environment.

III. System Model and Problem Formulation

In this section, we present satellite networks system and problem formulation.

A. System Model for Satellite Networks

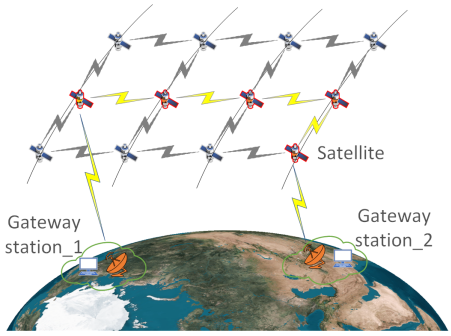


Fig. 1. Satellite networks system

As shown in Fig. 1, the reference system is consist of a LEO satellite constellation and a set of gateway stations. Gateway stations transmit data through the satellite network, and our goal is to find the optimal paths for data transmission in the LEO satellite constellation.

In this paper, a single-layer LEO polar satellite constellation is considered, where n_s satellites are distributed in n_o orbits. The satellite network can be modeled as a directed graph: $g = (V, E)$, where V is the set of satellites and E is the set of inter-satellite links (ISLs). Regularly, satellites have four ISLs, i.e., two intra-plane ISLs and two inter-plane ISLs. However, the inter-plane ISLs in the polar areas and the cross-seam ISLs cannot be built. To shield the topology changes caused by satellite movement, Virtual Node (VN) [4] is employed to abstract the satellite network. In VN, the logical position of a virtual node is used to represent the position of a satellite that virtual nodes and satellites correspond one to one at any time. When a satellite flies out of the range of the virtual node, the neighboring successive satellite will enter the position of this virtual node, which is called handoff. Handoff does not affect the topology for the permanent one-to-one correspondence but changes the links states. Based on VN, we can get a static topology with dynamic links states.

Note that what we are concerned about are the states of links, i.e., the attributes of edges in g, the state of the satellite network can be further denoted as:

$$s = (g, E_a) \tag{1}$$

where g shows the topology and E_a is the set of links states:

$$E_a = (e_1, e_2, ..., e_L) \tag{2}$$

where e_i is the state of link i and L is the total number of links in the satellite network. Besides, a link state contains a set of fields, i.e., $e_i = (f_1, f_2, f_3, f_4, f_5)$, where f_1 denotes the rest available capacity of the link, f_2 indicates the occupied bandwidth of the link, f_3 is the link betweenness (a measure of centrality inherited from graph theory that indicates how many paths may potentially traverse), f_4 is an action vector indicating whether the requested data is transmitted over this link, and f_5 is zero padding. Note that the dimension of a link state is related to how much information it encodes.

B. Problem Formulation

When a request arrives, we calculate k candidate paths (defined in (3)) using $k\_shortest\_paths$ algorithm [12]. Our aim is to find the optimal path from these candidate paths according to the current satellite network state.

$$\mathbf{P} = (p_1, p_2, ..., p_k) \tag{3}$$

If path p is selected as the optimal path, the available capacity of links in the selected path p can be defined as: $\mathbf{AC} = (ac_1, ac_2, ..., ac_{L_p})$, where L_p is the number of links in the selected path. The selected path should have enough available capacity to transmit the requested data:

$$bw \leq \min_{ac_l \in \mathbf{AC}} ac_l, \tag{4}$$

where bw is the required bandwidth.

As we know, transmission delay is key to Quality of Service (QoS). Here we define the transmission delay as the number of hops. In order to achieve a high throughput while keeping a low transmission delay, we construct a utility function for a request m using path p as:

$$U_m(bw_m, d_p) = U_{\alpha_1}(bw_m) - \lambda U_{\alpha_2}(d_p) \tag{5}$$

where bw_m is the bandwidth requirement of request m, d_p is the transmission delay of path p. Note that bw_m and d_p need to be normalized in advance. α_1, α_2 and λ are constants between 0 and 1. Usually, α_1 and α_2 can be set to the same value, λ is the weight factor that determines the importance of delay versus throughput. The functions $U_{\alpha_1}(bw_m)$ and $U_{\alpha_2}(d_p)$ have the following format:

$$U_\alpha(z) = \frac{z^{1-\alpha}}{1-\alpha} \tag{6}$$

Therefore, the optimization problem can be expressed as maximizing the utility:

$$\begin{aligned} &\max \sum_m U_m(bw_m, d_p) \\ &s.t. \quad (4). \end{aligned} \tag{7}$$

IV. The Proposed Grouting Algorithm

In this section, we present the proposed routing algorithm for satellite networks in detail.

A. Graph Neural Networks and Deep Reinforcement Learning

In this section, we introduce the basis of Graph Neural Networks (GNN) and Deep Reinforcement Learning (DRL).

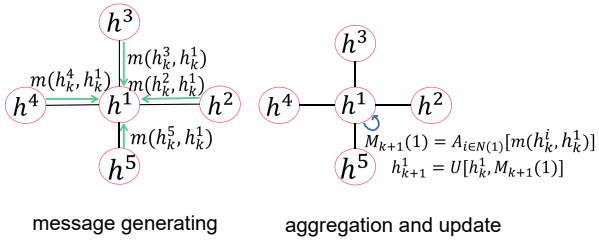


Fig. 2. Message passing over node 1.

1) Introduction of GNN: GNN is an emerging family of neural networks to deal with graph-structured data. GNN is a kind of representation learning, which is able to output the representation of graph-structured data. The core idea of GNN is that the state of each node in the graph is related to the state of its neighbor nodes. Thus all kinds of GNN variants try to explore the relationship between nodes and their neighbor nodes. Message Passing Neural Network (MPNN) [13] is the general framework of GNN that inputs some initial state of the graph and outputs the representation it learned. The forward propagation of MPNN can be divided into two phases: a message passing phase and a readout phase.

The message passing phase is a process of iterations, which can be defined by several functions:

- **Message function** $m(\cdot)$. Message function is used to generate messages from neighbor nodes for the center node.
- **Aggregation function** $A(\cdot)$. Aggregation function is used to aggregate all messages generated by message function.
- **Update function** $U(\cdot)$. Update function is used to update the hidden state of the center node with its previous hidden state and the aggregated message.

Then, the message passing process shown in Fig 2 can be represented as:

$$M_{k+1}(o) = A_{i \in N(o)}[m(h_k^i, h_k^o)] \tag{8}$$

$$h_{k+1}^o = U[h_k^o, M_{k+1}(o)] \tag{9}$$

where $i \in N(o)$ means that node i is the neighbor node of node o. $M_{k+1}(o)$ is the aggregated message for node o in iteration $k+1$. h_0^o denotes the initial hidden state of node o. Once the initial state is given, every node o can updates its state h_k^o according to (8) and (9). The iteration process will go on for K times to get a fixed point representation h_K^o for every node o.

After iterating K times message passing, a **readout function** $R(\cdot)$ is used to output the representation hp in readout phase:

$$hp = R(\{h_K^o | o \in g\}) \tag{10}$$

In MPNN, message function, update function and readout function can be learned by neural networks. Generally, the message function and readout function can be approximated by fully-connected neural networks, while the update function can be implemented with recurrent neural networks (RNN).

2) The basis of DRL: Based on the description in Section III, the state space, action space and reward function for the DRL framework can be defined as follows:

- **State space**: In our algorithm, the DRL agent chooses actions based on the state representation learned via GNN. Thus, the state space is consist of all the possible value of hp, so we have $\mathbf{S} = \{hp_1, hp_2, ...\}$.
- **Action space**: The candidate paths **P** for all possible source-destination pairs in the satellite network form the action space **A** .
- **Reward function**: The reward function indicates the immediate reward that the DRL agent can get from the environment after it takes a certain action. Here we use the utility function as the reward function:

$$r = U_m(bw_m, d_p) \tag{11}$$

In this paper, Deep Q-Network (DQN) [14] is employed to learn the optimal routing policy for the LEO satellite network, which is extended from Q-learning by utilizing neural networks. In DQN, the Q-value is defined as the expected cumulative reward that evaluates how good the chosen action is for its corresponding state:

$$Q(s,\ a) = E[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t) | s_0 = s, a_0 = a] \tag{12}$$

where γ is a discount factor. According to Bellman equation, the Q-value function can be expressed as:

$$Q(s_t, a_t) = r(s_t, a_t) + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}) \tag{13}$$

Then, the action that provides the maximum Q-value for state s_t will be chosen:

$$a_t = \arg\max_{a' \in \mathbf{A}} Q(s_t,\ a') \tag{14}$$

DQN [14] uses a neural network to approximate the Q-value. In DQN, two neural networks $Q(s, a; \theta)$ and $Q(s, a; \theta^-)$ are preserved, where the θ and θ^- are the network weight parameters. $Q(s, a; \theta)$ is the evaluation network for selecting an action while $Q(s, a; \theta^-)$ is the target network for training. In addition, the transition $\langle s_t, a_t, r(s_t, a_t), s_{t+1} \rangle$ will be stored into an experience replay pool D, then the Q network will be updated with mini-batch sampled from D. In addition, the loss of DQN is defined as follows:

$$\begin{aligned} L(\theta) = E_{(s,a,r,s') \in D}[r(s,a) + \gamma \max_{a'} Q(s', a'; \theta^-) \\ - Q(s, a; \theta)]^2 \end{aligned} \tag{15}$$

where $(s, a, r, s^{'})$ is the transition randomly sampled from D. Then, the gradient can be calculated as:

$$\begin{aligned} \nabla_\theta L(\theta) = E_{(s,a,r,s') \in D}[(r(s,a) + \gamma \max_{a'} Q(s', a'; \theta^-) \\ - Q(s, a; \theta)) \nabla_\theta Q(s, a; \theta)] \end{aligned} \tag{16}$$

B. Graph-DQN based routing algorithm

According to the aforementioned, we design the GNN-DRL based network architecture for our proposed GRouting algorithm, as shown in Fig 3. The message function and update function are approximated by a three-layer fully-connected and a one-layer recurrent neural network respectively. The default aggregation function is set as the $\sum$ function and the readout function is approximated by a three-layer fully-connected neural network. The Q-Network is a five-layer fully-connected neural network. This algorithm contains two parts: the training process and test process. Algorithm 2 has shown the training process and the test can be done through replacing the DQN agent with the trained agent. Note that, $gnn.propagate()$ in line 15 is the forward propagation described before and its detail is shown in Algorithm 1. Because the input satellite networks state is basically composed of topology and the links states, it will first be transformed into its corresponding linegraph, which is able to convert edges into nodes while ensuring the equivalence. Even though handoff happens often in Virtual Node-based satellite networks, the state after a handoff can be directly sent to our model to get routing paths once our model is trained completely for GNN's generalization.

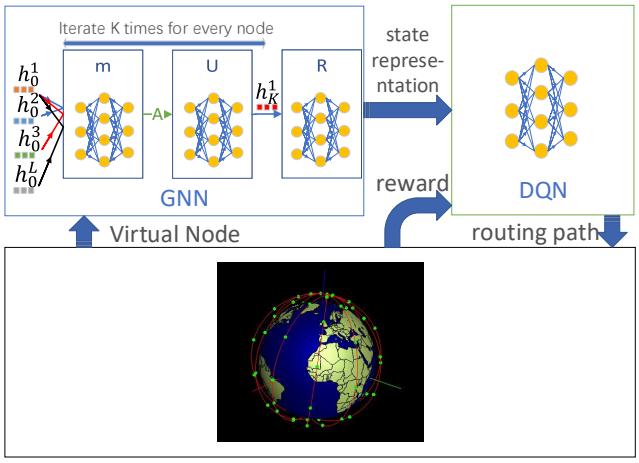


Fig. 3. Architecture of proposed GRouting algorithm for LEO satellite networks.

Algorithm 1 MPNN on satellite networks

1: Input satellite networks state s with link initialization
2: $s_l = s.linegraph()$
3: **for** k in range(K) **do**
4: **for** every node $o \in s_l.g$ **do**
5: $M_{k+1}(o) = A_{i \in N(o)}[m(h_k^i, h_k^o)]$
6: $h_{k+1}^o = U[h_k^o, M_{k+1}(o)]$
7: **end for**
8: **end for**
9: $hp = R(\{h_K^o | o \in s_l.g\})$

V. Performance Evaluation

A. Experiment Setup

In this paper, we use the Iridium constellation as the target LEO satellite network, where 66 satellites are distributed in 6 orbits and we select 6 of these satellites as the source and destination nodes. For each transmission, the source and the destination satellites are chosen randomly from the selected 6 satellites and the required bandwidth is randomly generated from several kinds of packages (i.e., 16, 32, 64 bandwidth units). Once a request is implemented, the occupied bandwidth will not

Algorithm 2 Routing Algorithm with Graph-based DQN

```
s, g, req = env.init()
DONE = False, reward = 0, k
gnn.init():
    initialize the GNN model with ω
agt.init():
    agt.rmb_counter, agt.memory_capacity
    randomly initialize the evaluation network with θ
    initialize the target network with θ⁻ = θ
while not DONE do:
    k_paths = k_shortest_paths(g, req.src, req.dst, k)
    k_qvalues = {}
    k_sprime = {}
    for i in range(k) do
        k_sprime[i]= env.apply_req(s, req.bw, k_paths[i])
        k_sprime[i]= gnn.propagate(g, k_sprime[i])
        k_qvalues[i]= agt.eval_net(k_sprime[i])
    end for
    action = agt.choose_action(k_qvalues, ϵ)
    r, DONE, s', req' = env.step(s, action)
    agt.store_transition(s, action, r, s')
    agt.rmb_counter += 1
    reward += r
    req = req'
    if agt.rmb_counter >= agt.memory_capacity then
        agt.experience_replay():
            sample a minibatch
            calculate the loss and gradient
            update the weights ω and θ
        agt.learning_step += 1
    end if
    if agt.learning_step % M == 0 then
        θ⁻ = θ
    end if
end while
```

be freed until the episode ends and an episode ends when (4) is not satisfied. The initial bandwidth for every link is set to be 200 bandwidth units. The exploration rate ϵ decays from 1 exponentially every 3 episodes and stops at 0.001. We use a tuple to denote the number of neurons for neural networks, where the first element and the last element denote the number of neurons for the input layer and output layer respectively, and the other elements are the number of neurons for hidden layers. Then, (40, 20, 20), (4240, 2048, 1024) and (1024, 256, 64, 16, 1) can be used to represent the number of neurons for message function, readout function and the Q-Network. As to the update function, the hidden size of RNN is set to be 20. The setting of other parameters is shown in Table I.

B. Training Process

In order to illustrate the convergence of our model, we trained our model for 400 episodes and the training process is shown in Fig. 4. Note that the reward in Fig. 4 is the average reward for every 10 episodes.

As we can see, the DQN agent was consistently trying to find the optimal routing paths before about the first 200 episodes. After 200 episodes, the average reward is becoming converged, which means that the DQN agent has finished the learning process and can find the optimal paths.

C. Baseline Algorithms

In order to compare our GRouting algorithm with some baseline algorithms, three widely used baseline solutions and a method using DQN without GNN that are also based on Virtual Node are selected:

- Shortest Path (SP). SP chooses the shortest path for every coming request.
- Random Path (RP). RP decides the routing path randomly from the candidate paths.
- Request Balance (RB). RB evenly splits the required bandwidth into k parts and distributes them to all k candidate paths.
- DQN without GNN (DWG) [15]. DWG utilizes DQN algorithm without GNN, i.e., it directly uses a fully connected neural network to approximate the Q-value of satellite networks state.

TABLE I
CONFIGURATION AND EXPLANATION FOR PARAMETERS

Parameters	Explanation	Values
$\alpha_1, \alpha_2, \lambda$	parameters for reward	0.9, 0.9, 1
K	iterative number of GNN	12
k	number of candidate paths	5
n	dimension of link state	20
lr	learning rate	0.0001
γ	discount factor	0.9
D	replay pool size	3000

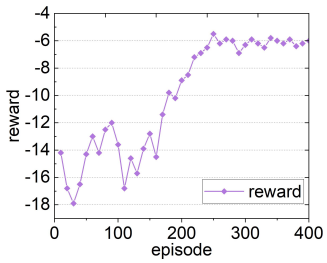


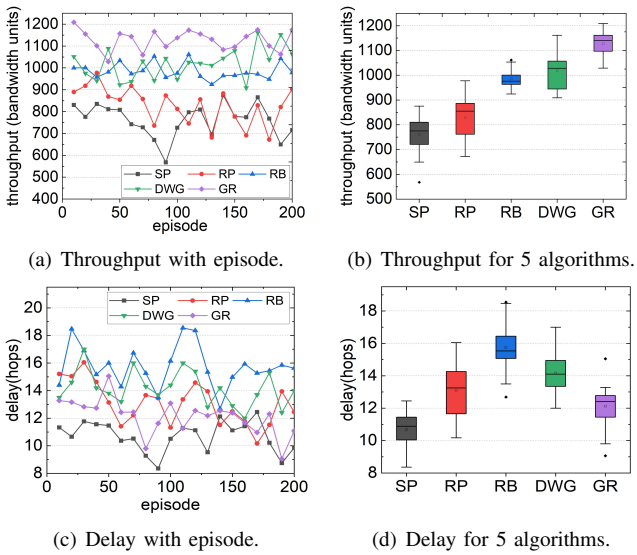
Fig. 4. Convergence analysis for GRouting.

D. Performance Comparison

We compare GRouting with four baseline algorithms in terms of *network throughput* and *transmission delay*. We test 200 episodes for our trained model and baseline algorithms with the same sequence of requests and the same number of candidate paths. The experiment results are shown in Fig. 5 and we can see:

1) Compared with SP algorithm, our algorithm outperforms amazingly in throughput with comparable delay. Because SP algorithm always chooses the shortest paths for requests, an episode is quick to end and the delay is short. The throughput has a 47.1% improvement with a 13.7% delay increase for our algorithm.
2) In comparison with RP, our algorithm outperforms not only in throughput but also in delay. Our algorithm improves the throughput by 33.4% and decreases the delay by 6.1%.
3) Compared to RB, our algorithm has a much better performance in throughput and delay. RB obviously increases the throughput compared with SP and RP, however, the delay also increases. Our model gets a 17.5% higher throughput and 20.1% lower delay than RB.

4) Campared with DWG, our model achieves better performance in throughput and delay. This is because DWG doesnot have generalization ability. Numerically, our model increases the throughput by 10.7% and decreases the delay by 12.1%.



(a) Throughput with episode. (b) Throughput for 5 algorithms.

(c) Delay with episode. (d) Delay for 5 algorithms.

Fig. 5. Performance comparison with four baseline algorithms.

E. Performance Sensitivity Analysis

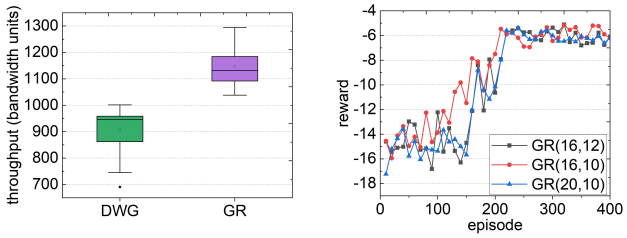
In this section, we conduct more experiments to 1) illustrate GNN's generalization and 2) analyse the performance with different parameters.

First, we change the topology of satellite networks, i.e., one satellite is removed. Then, the trained DWG model and our model are tested for 200 episodes in terms of throughput. The result is shown in Fig. 6(a). It's obvious that DWG has a worser performance on the new topology, however, our model can still get a comparable throughput with different topologies, which directly demonstrates GNN's generalization ability.

Second, Fig. 6(b) shows training process with different parameters. Note that, $GR(n, K)$ in Fig. 6(b) represents that our proposed algorithm employes n dimensional link states and K MPNN iterations. It can be found that different parameters result in almost the same converged reward, which means that all of them are able to learn the optimal routing policy. However, we can see that the larger link state dimension and iterative number, the more episodes are needed for training the model.

VI. Conclusion

In this paper, we present a Graph-based DQN routing algorithm for satellite networks combining Virtual Node. First, we describe the scenario and build the satellite networks system by Virtual Node. Then, the emerging GNN model is used to learn the representation for dynamic satellite networks states. In addition, GRouting is proposed to choose the optimal routing paths for satellite networks via Graph-based DQN. Finally, extensive experiments are conducted to illustrate that: 1) GNN has great generalization ability that enables our model to capture the dynamics of the LEO satellite networks (including topology, links states and workload/congestion); 2) GRouting is able to increase the throughput at most 47.1% and decrease the transmission delay at most 20.1% compared with the baseline algorithms.



(a) Throughput over a new topology. (b) Training process with different parameters.

Fig. 6. Performance analysis with a new topology and different parameters.

References

[1] B. Di, L. Song, Y. Li, and H. V. Poor, "Ultra-dense leo: Integration of satellite access networks into 5g and beyond," *IEEE Wireless Communications*, vol. 26, no. 2, pp. 62–69, 2019.

[2] M. Werner, "A dynamic routing concept for atm-based satellite personal communication networks," vol. 15, pp. 1636–1648, 1997.

[3] M. Jia, S. Zhu, L. Wang, Q. Guo, H. Wang, and Z. Liu, "Routing algorithm with virtual topology toward to huge numbers of leo mobile satellite network based on sdn," *Mobile Networks and Applications*, vol. 23, no. 2, pp. 285–300, 2018.

[4] Q. Chen, J. Guo, L. Yang, X. Liu, and X. Chen, "Topology virtualization and dynamics shielding method for leo satellite networks," *IEEE Communications Letters*, vol. 24, no. 2, pp. 433–437, 2019.

[5] C. Wang, H. Wang, and W. Wang, "A two-hops state-aware routing strategy based on deep reinforcement learning for leo satellite networks," *Electronics*, vol. 8, no. 9, p. 920, 2019.

[6] J. Li, H. Lu, K. Xue, and Y. Zhang, "Temporal netgrid model-based dynamic routing in large-scale small satellite networks," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 6, pp. 6009–6021, 2019.

[7] Z. Liu, J. Li, Y. Wang, X. Li, and S. Chen, "Hgl: A hybrid global-local load balancing routing scheme for the internet of things through satellite networks," vol. 13, p. 155014771769258, 2017.

[8] H. Wang, G. Wen, N. Liu, J. Zhang, and Y. Tao, "A load balanced routing algorithm based on congestion prediction for leo satellite networks," *Cluster Computing*, vol. 22, no. 4, pp. 8025–8033, 2019.

[9] W. Sun, J. Liang, N. Xiao, R. Ding, and Z. Zhang, "Intelligent routing scheme for sdn satellite network based on neural network," in *IOP Conference Series: Materials Science and Engineering*, vol. 563, no. 5, p. 052087, 2019.

[10] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," vol. 20, pp. 61–80, 2009.

[11] J. Li, H. Lu, K. Xue, and Y. Zhang, "Temporal netgrid model-based dynamic routing in large-scale small satellite networks," vol. 68, pp. 6009–6021, 2019.

[12] B. Y. Chen, X.-W. Chen, H.-P. Chen, and W. H. K. Lam, "Efficient algorithm for finding k shortest paths based on re-optimization technique," vol. 133, p. 101819, 2020.

[13] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, "Neural message passing for quantum chemistry," Apr. 2017.

[14] V. Mnih *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[15] P. Almasan *et al.*, "Deep reinforcement learning meets graph neural networks: exploring a routing optimization use case," *arXiv*, pp. arXiv–1910, 2019.