SYSC 4700-A, Winter 2026
Topics in Communications Networks

Final Project Report

Professor: ***Afsoon Alidadi Shamsabadi***

Submitted by: Samuel Burt 101260404,
Colin Byrne 101224212,
Earnest George 101212512

Group: 6

Date Submitted: 04/12/26 *(MM/DD/YY)*

Carleton University

Table of Contents:

I. Abstract

Satellite systems are a constantly evolving industry that allow near-seamless communication from opposite ends of the globe. As advancements are made in this discipline, older rule-based technologies often become obsolete, needing to be replaced. To extend the service life of these satellites, these devices have been taught to adapt to their surroundings through reinforcement learning (RL).

Due to the ever-changing wireless traffic demand throughout the day, numerous satellite systems, notably Very High Throughput Satellite (VHTS) systems, will experience nonuniform increases in traffic demand. This work provides an overview of recent and future developments in reinforcement learning for satellite communications that address this issue, and many more.

The structure of this paper will give an introduction into HTS and VHTS systems and why reinforcement learning was chosen to look into. Then it will go in-depth as to what reinforcement learning is, what neural networks are, the equations behind reinforcement learning, and what deep Q-networks are. Then we will look more in-depth into what adaptive resource management is, what multibeam HTS means, and what current methods are being researched into this topic. Lastly, we discuss the issues that come with working in such complex systems and how AI algorithms can be integrated into these satellite systems.

II. Introduction

Satellite systems are categorized by their altitude and function, LEO (low-latency internet/imaging), MEO (navigation/GPS), and GEO (broadcasting/weather). The GEO stations are typically further than 35,000 kilometers away from Earth and typically work with a conventional satellite system as seen in Figure 1. This gives it a limited throughput of about 1-10 Gbps. Both MEO and LEO stations typically use the multibeam HTS system to give it a narrower band and higher throughput. The VHTS system, or EHTS system as seen in Figure 1, is a future concept as problems are still being solved in HTS.

High Throughput Satellites (HTS) are an advancement over conventional satellite systems that significantly increase data capacity by utilizing frequency reuse and spot beam technology. Unlike traditional wide-beam GEO satellites that cover large geographic regions with a single beam, HTS systems divide the coverage area into multiple smaller spot beams, each capable of operating on the same frequency bands without causing interference. This spatial separation allows for a substantial increase in overall system throughput [2].

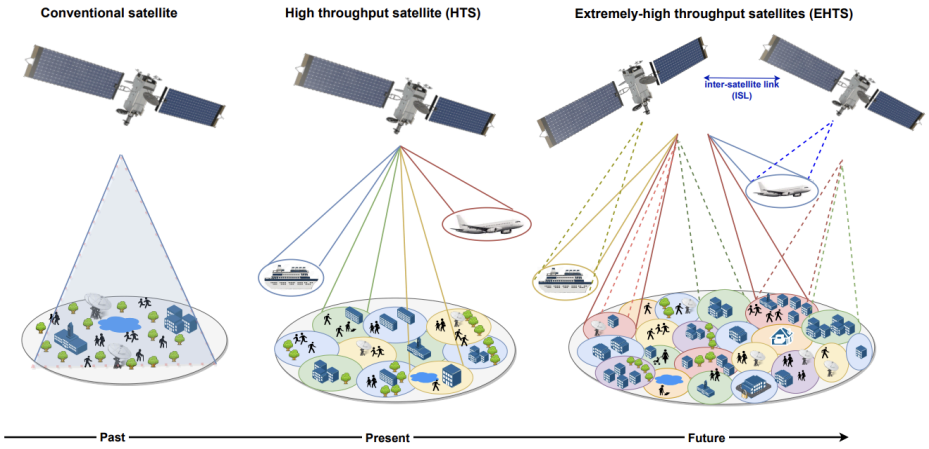

Figure 1: Evolution of SatCom from single-beam satellite to VHTS [1]

HTS systems typically operate in higher frequency bands such as the Ka-band, enabling wider bandwidth allocations and higher data rates. The satellite payload includes multiple transponders connected to these spot beams, which are managed through advanced onboard processing or bent-pipe architectures [3]. Ground stations communicate with the satellite using gateway hubs that aggregate traffic and distribute it across the network. By combining spot beam architecture with frequency reuse and higher frequency spectrum, HTS systems can achieve throughput levels far exceeding those of conventional satellite systems, often reaching tens or even hundreds of Gbps depending on system design [4].

For next-generation networks, resource management is a vital factor in the systems, and the complexity needed to handle this task is far too complicated for current algorithms to work. Deep reinforcement learning is a growing practice that can be used to tackle the tasks of power control and bandwidth allocation. DRL is currently being applied to advanced satellite systems for rate-splitting multiple access (RSMA) in LEO constellations. This example shows the potential of DRL, and now there are plenty of papers and research into the possible DRL frameworks to address the issues in multi-beam systems [5].

In general, resource management in satellite networks involves multiple interdependent decisions, such as power distribution, bandwidth allocation, and beam scheduling, all of which must adapt to highly dynamic environments. DRL shows significant promise for these sequential decision-making problems, with early studies demonstrating its effectiveness for general resource management in complex networks and directly tackling dynamic allocation in multibeam satellite systems. More recent work has extended these approaches to multi-objective DRL strategies for time-frequency allocation, energy efficiency optimization, and guaranteeing diverse quality-of-service levels, while also exploring applications in routing, computation offloading, and large-scale LEO constellations [5]. Despite this progress, few studies have addressed simultaneous optimization across multiple resources under realistic operational constraints, highlighting a clear gap. Bridging this gap could provide more robust, adaptive, and

intelligent resource management frameworks that will be critical for the next generation of satellite networks.

III. Reinforcement Learning

III.a. Reinforcement Learning Framework

Reinforcement learning has four main parts to its system: a policy, a reward signal, a value function, and a model of the environment.

Policy: The policy is the map connecting the states of the environment to the actions taken, this can be seen as the policy being the behaviour of the agent as it learns. In Q-Learning, the policy is represented as a simple lookup table, and in some cases, it can even be a simple function. However, in more complex systems, such as DQN, the policy is a neural network with weights and biases.

Reward Signal: The reward signal is the definition of the problem, it is what the environment sends to the agent after every step. It is what defines a good or bad action taken, with the goal of the agent to be maximizing the total rewards. The rewards are what determine the policy, if the reward is positive then the policy will remember to possibly take that action in the future, whereas if the reward is negative then the policy will avoid that situation in the future.

Value Function: The value function differs from the reward in the sense that it looks into the long run of the system. It finds the total accumulated rewards possible from the current state, such as a state may give immediate low reward, but have much higher rewards in states that can only be achieved by taking the initial low reward. Since values are still dependent on rewards, they come second to rewards, yet actions are based on value judgments. Value estimation is a crucial aspect of an agent, as rewards are provided by the environment, and developing methods to estimate these rewards has been a key focus of reinforcement learning.

Model of the environment: The model of the environment is to mimic the behaviour of real-world environments, and what allows for planning. Models are optional as there are model-based methods as well as model-free methods which work with trial-and-error learners.

A common approach at the value estimation is the use of the *Epsilon-Greedy* policy which uses a parameter of Ɛ. Ɛ is typically between 0 and 1, and it makes up a percentage of whether the agent will take a random action or choose the highest value action. For example, if Ɛ is 0.1, then there is a 10% chance the agent will explore randomly, and a 90% chance the agent will choose the highest value. The purpose of the method is to avoid choosing a suboptimal pathway.

An example of how this method is used is seen in the code snippet below:

```
epsilon = 1.0
epsilon_min = 0.05
```

```
epsilon_decay = 0.995

for ep in range(episodes):
    state, _ = env.reset()
    done = False
    total_reward = 0

    while not done:
    # epsilon-greedy
    if random.random() < epsilon:
        action = env.action_space.sample()
    else:
        state_tensor = torch.tensor(state, dtype=torch.float32, device=device).unsqueeze(0)
        with torch.no_grad():
            q_vals = policy_net(state_tensor)
            action = q_vals.argmax().item()
    ...
epsilon = max(epsilon * epsilon_decay, epsilon_min)
```

In the example provided, Ɛ starts at 1, and at the end of the episode, Ɛ is shrunk by a multiple of 0.995, until it reaches 0.05. In the while loop, a random number between 0 and 1 is generated and compared to epsilon, if it is smaller, then the action taken is a random action space sample. Starting at 1 and slowly decreasing means that early on it is often random spaces to help learn. This also means that even when epsilon is small, the action taken on the next step could still be random. This allows for the agent to constantly be learning until the agent is ready to be tested and Ɛ-greedy is removed.

III.b. Bellman Equation

The Bellman equation is a recursive equation that represents the value function of a state. It can be broken down into five separate components to create the equation, as seen in Eq. 1.

$$V^{\pi}(s) = \sum \pi(a|s) \sum P(s'|s,a)[R(s,a,s') + \gamma V^{\pi}(s')] \quad [6]\ (1)$$

The value function of the state, $V^{\pi}(s)$, as mentioned previously, is broken up into five components. The first is the sum of the probability of taking action a in state s, $\pi(a|s)$. This part reflects the average over all possible actions that may be taken by the agent. Then it is the sum of the probability of ending up in the next state s', $P(s'|s,a)$. This is to find the average over all possible next states. Then inside the bracket, we have the immediate reward for going from state s to state s' with action a, $R(s,a,s')$ and lastly we have the discount rate, γ, which controls how much future rewards matter. *Discounting* is an important concept as it determines the present worth of future rewards, this is to ensure that the agent receives higher value for a reward three steps away rather than a reward of equivalent worth ten steps away. A good visualization of discounting is with the *discounted return* seen in Equation 2 below [6].

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \quad \bullet\bullet\bullet \quad = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \quad [6]\ (2)$$

In Eq. 2, t represents time steps, G_t is the return, and k is the number of time steps taken. This equation shows that a reward received k times steps in the future is scaled down by the discount factor, γ^{k-1}, over time, making it less valuable than an immediate reward. It prevents the agent from being too farsighted or too short-sighted [6]. This discount rate is then applied to future value functions, $V^{\pi}(s')$, for the stated reasons. Overall, the Bellman equation captures the idea that a state's value is determined by averaging over possible actions and future outcomes.

III.c. Q-Learning

Q-Learning is an algorithm used in reinforcement learning that incorporates values called Q-values and treats the action and states as a table. The algorithm changes the Bellman equation and replaces the value function with the Q-value, the equation can be seen below as Equation 3.

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[r + \gamma max_a Q(S_{t+1}, a) - Q(S_t, A_t)] \quad [6](3)$$

This equation uses a *Temporal Difference (TD)* update rule and the values found represent the expected rewards taken from action A_t in state S_t, $Q(S_t, A_t)$, much like that in the Bellman equation. The reward in Eq. 3 is represented by *r,* the discount factor is the same, γ, and α is the learning rate. The learning rate is a step-size parameter that is a constant value, yet acts as if it changes from time step to time step. This change comes from the use of old steps in order to minimize past rewards, and similar to the discount factor, there is an equation that represents the use of the learning rate:

$$\begin{aligned} Q_k &= Q_k + \alpha[R_k - Q_k] \\ &= (1 - \alpha)^k Q_1 + \sum_{i=1}^{k} \alpha(1 - \alpha)^{k-i} R_i \end{aligned} \quad [6](4)$$

Equation 4 is called the *weighted average,* as the weight of the given reward, R_i, depends long ago the reward was observed, $k - i$. The learning rate is sometimes better as a varying parameter from step to step, $\alpha_k(a)$, as it can increase when change is detected and decrease when it is more stable [6]. This means that a dynamic learning rate is better for adapting to a greatly changing environment.

As mentioned, the Q-learning algorithm uses a table called the Q-table, which the agent initializes with random values. From this table, using the aforementioned Ɛ-Greedy method, the agent chooses and performs the action. The agent then observes the new state and the immediate reward given and uses Eq. 3 to update the Q-table. This cycle is repeated until the agent is told to stop and the table is complete, giving the agent the optimal policy.

Overall, Q-learning is an off-policy reinforcement learning algorithm, meaning it learns the optimal policy regardless of the agent's exploration strategy. It balances exploration and exploitation through methods such asε-greedy. While Q-learning is guaranteed to converge under certain conditions, such as sufficient exploration and appropriate learning rate decay, it suffers from scalability issues in large state spaces [6].

An example of Q-Learning in Python is seen below. The first step is the initialization of the Q-Table, in this code, the Q-Table is initialized with all zeroes instead of random values:

```
q = np.zeros((env.observation_space.n, env.action_space.n))
```

Then the learning rate and discount rate are initialized:

```
learning_rate_a = 0.9
discount_factor_g = 0.9
```

Then the epsilon Ɛ-Greedy method is initialized, similar to how it was initialized in the previous example. Lastly, we enter a while-loop cascaded inside of a for-loop where we use Eq. 3 after every step:

```
   for episode in range(episodes):
        print(episode)
        state = env.reset()[0]
        terminated=False
        truncated=False


        while(not terminated and not truncated):
            if is_training and rng.random() < epsilon:
                action = env.action_space.sample()
            else:
                action = np.argmax(q[state,:])


            n_state,reward,terminated,truncated,info = env.step(action)


            if is_training:
                q[state,action] = q[state,action] + learning_rate_a * (
                    reward + discount_factor_g * np.max(q[n_state,:]) -
q[state,action]
                )
            state = n_state
```

```
        epsilon = max(epsilon - epsilon_decay_rate, 0)

        if(epsilon==0):
            learning_rate_a = 0.0001

        if reward==1:
            rewards_per_episode[episode] = 1
```

III.d. Deep Q-Learning

To address the issue of Q-Learning, neural networks are used instead of the Q-table, this is referred to as a Deep Q-Network (DQN). Neural networks are a network of units, with each unit representing a neuron, with weights and a bias attached to each neuron. A way to look at it is that reinforcement learning is meant to replicate human decision-making with a reward system, and a neural network is meant to replicate the human neural network. The human neuron is what allows people to communicate with themselves through electromechanical signals. These neurons travel through the nervous system and are ultimately perceived as sensations like movement or pain [7]. Neural networks, such as those in DQN, recreate these complex systems in software using Equation 5 below and represented as Figure 2 below.

$$z = h(w^T x + b) \quad (5)[8]$$

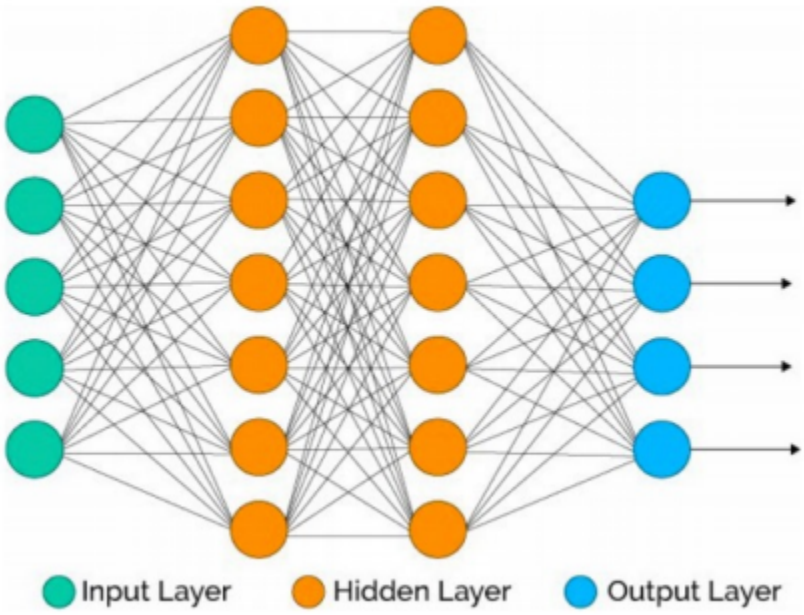

Figure 2: Visual representation of a neural network [8]

The matrix of weight is represented as w^T in Eq. 5 and the lines in Fig. 2, the inputs are represented by x in Eq. 5 and the input and hidden layers in Fig. 2, and lastly, the output is

represented as z in Eq. 5 and the output and hidden layers in Fig. 2. The two variables not seen in the figure are the activation function, h, and the bias b. The activation function determines a neuron's output by introducing non-linearity, allowing neural networks to learn complex patterns. It is commonly a Rectified Linear Unit (ReLU), sigmoid, tanh, or Gaussian. The bias is an extra parameter added to a neuron that allows it to shift the activation function, helping the model make better predictions even when all inputs are zero [8].

DQN uses a technique called experience replay, where past experiences are stored in a buffer and a mini-batch of these experiences is randomly sampled at each step to train the model using Q-learning. Another technique used by DQN is having two networks, the target and the main, which uses the target network to provide stable target Q-values during training. With this, the target network is only updated periodically, while the main network is actively learning from new data. From Eq. 3, we can get the equation for the weights of the mini-batches:

$$w \leftarrow w - \alpha[Q_w(s,a) - r - \gamma max_{a'} Q_{\overline{w}}(s',a')]\Delta Q_w(s,a;w) \qquad (6)[8]$$

Where $Q_{\overline{w}}(s',a')$ is the target network, and $Q_w(s,a)$ is the main network, then after a chosen number of sets, the weights of main network are copied into the target network, $\overline{w} \leftarrow w$. The gradient, $\Delta Q_w(s,a;w)$, measures how the Q-value changes with respect to the network's weights and is used to update the model during learning. Overall, DQN combines reinforcement learning with deep neural networks to approximate optimal policies in high-dimensional environments where traditional Q-learning is not capable [8].

IV. Satellite Communications

IV.a. Resource Management in VHTS Networks

In order to increase overall system capacity, VHTS systems use multibeam architectures and spatial reuse. Even though this may seem advantageous at first glance, it creates a challenge when resource management is factored into consideration. This is because of the uneven distribution of traffic across beams that change with respect to time. Various resources that include spectrum, power, beam time, and routing paths are dynamic in nature and hence must be allocated efficiently. In a real-world scenario, there must be a balance between multiple key performance indicators (KPIs) that are subject to the controller [9]. These KPIs include throughput, delay, fairness, quality of experience, and power efficiency. Therefore, resource allocation in VHTS systems is a multi-objective and dynamic control problem and should not be treated as a static one.

IV.b. Current Resource-Management Methods and Their Limitations

Current resource management techniques in satellites often rely on rule-based scheduling, optimization-based control, threshold heuristics, or static beam provisioning. The currently employed approaches are used because of their simplicity, interpretability, and stability under predictable operating environments. However, in an advanced technological era where data transfer is at rapid speeds, their limitations are exposed. To illustrate, rule-based methods under dynamically changing traffic conditions can be brittle [10]. There is huge model dependence or high computational cost when optimization-based approaches are adopted. Long term effects can be ignored by threshold heuristics and static provisioning is inefficient when traffic distribution is uneven across the multibeam architecture. Therefore, these methods suffer practicality when the network is dynamic and ever-changing.

IV.c. Reinforcement Learning as a Possible Solution

Because of the sequential nature of reinforcement learning (RL), it is well suited for VHTS resource management, especially in dynamic networks. RL has the ability to learn policies through constant interaction and improve the system's long term performance through optimization techniques rather than just focusing on the immediate conditions. This capability is not seen in static methods, which is a major drawback. This is key and one of the major advantages of RL because future queue lengths, congestion, delay, and service quality are directly affected by current resource-allocation decisions. In the context of satellites, RL can be used for learning adaptive policies that support bandwidth allocation, beam scheduling, power control, and traffic steering while ensuring there is a balance between the KPIs such as throughput, delay, QoE, and power efficiency [10].

IV.d. MDP Formulation of Satellite Resource Allocation

The resource allocation model in satellites can take inspiration from one of the frameworks of the Markov Decision Process. In this formulation, the satellite network controller is the agent and the state may be represented as beam demand, queue length, RTT, throughput, and available spectrum or power. The actions that could be taken include bandwidth allocation, traffic steering, beam scheduling and power control. The rewards modelled behind these factors are designed to reflect the tradeoff between throughput, delay, congestion, QoE (Quality of Experience), and power use [11]. This formulation is key because of its ability to capture the sequential aspect of the problem. To sum, the action taken in the current time step reflects changes for network states in the future thereby improving long term performance.

V. Challenges and Open Research Issues

There are several challenges that remain to be addressed in the possible application for RL in VHTS systems. Firstly, networks and traffic dynamics are nonstationary, meaning training of

one policy for a specific pattern may not be applicable to another. Hence, each policy is quite unique and is not generalised enough. Secondly, the reward design could be fragile because of feeding in a specific KPI which strongly influences the entire model. This is a major concern since resource management is multi-objective in nature and hence requires a balance among KPIs. Thirdly, scalability is a concern because state spaces are vast and hybrid action spaces require both continuous and discrete controls. Lastly, good performance cannot be guaranteed by simulation alone because it does not reflect real satellite environments which raises questions about deployment.. Future research work should focus on matching the RL method, reward structure, and evaluation setting to a control task that is specific to a satellite.

VI. Suggested Novelty

VI.a. Reinforcement Learning in Very-High Throughput Satellites (VHTS)

Reinforcement learning models nominally utilize a feed-forward topology. This means it takes past training data to constantly update its model, though the model is never exactly perfect during the time the update is needed. This is because it will always be one update cycle behind real-time, due to satellite systems being a constantly changing environment.

To fix this, we can utilize deep Q-learning's prediction (target) network function to get one step ahead of these updates. By setting this target network's Q-value table to estimate the channel's future CSI (by predicting an event), the feed-forward topology should suddenly work in real-time, as opposed to being behind the update cycle. This predictive network would likely need its own reinforcement model to make accurate event predictions, but ultimately, unless under extreme events, the system should stabilize and work effectively in real-time [14].

This design would work well when paired with the satellites' Multi-In Multi-Out (MIMO) systems, due to their involvement with most CSI parameters. For example, MIMOs would see spectrum allocation, signal power, beam time, routing paths, latency, path loss, doppler shifts, channel fading, and many more parameters pertinent to the channel state information of the satellite's wireless links, ultimately having the ability to place all of these actions and states into a Q-table. Then, after iterative learning, the environment should pose little threat to the system due to its preparedness for all known factors, or 'states'. The deep Q-learning network would evaluate the available decisions and pick the best action to take based on this state. For example, a satellite constellation suffering path loss or doppler shift should ideally result in rerouting, or stronger signal strength to reach neighbouring satellites– a decision made by the deep Q-learning network.

VI.b. Hardware Implementations

Deep reinforcement learning algorithms are highly complex, and therefore require heavy computational power to execute quickly. Even with the added predicted state as mentioned above, the latency can suffer so greatly that the updates fall behind regardless. This is a problem

for hardware implementation, making sure that the throughput of these model updates are as efficient as possible.

Neural network computations get exponentially more computationally expensive with more parameters, but ultimately the math is the same. Matrix multiplications are a long chain of multiplying, adding, and doing it over and over again. A piece of hardware that comes to mind for these repeated processes is the Field-Programmable Gate Array (FPGA), a type of circuit board renowned for its ability to have various processes in parallel, as long as the board contains enough of the module. Since it would only have to house two different modules (and likely some overhead), an FPGA custom-built for multiplier-adder circuits could ideally run thousands, if not logarithmically more of these neural network matrix multiplications all at once, serially [12].

VI.c. Extremely-High Frequency (EHF) Band

Currently, the primary frequency band for satellite communications is in the Ka band (26.5-40 GHz). With the ever-increasing number of satellites in orbit, the Ka band spectrum is becoming a scarce resource. In the future, the EHF band will emerge as the predominant frequency range for satellite systems once engineered. This offers not only a plentiful spectrum, from 30 GHz to 300 GHz, but also the benefits of transmitting data at that rate. With so much data being fed through these deep reinforcement learning models, higher frequencies are a must-have for lower latency, real-time systems [13].

This is evidently limited by currently available hardware, requiring high-power transmitters and receivers, as well as adaptive antennae. Fortunately, when this technology does arise, satellite systems should be one of the prime targets for its implementation, since these satellites are travelling through Earth's orbit. In low-earth orbit the medium becomes ideal for electromagnetic waves, due to its near-vacuum-like conditions. When these EHF components are developed, it would also heavily benefit these satellite systems to transmit data through optics, due to constellations' almost entirely uninterrupted line-of-sight between nodes.

VI.d. DQN Agent

The design of a DQN agent to be able to work with VHTS systems' path loss requires an experience replay, as mentioned in Section *III.d. Deep Q-Learning*. The environment changes when satellites fail, so replay helps the agent learn across multiple network configurations instead of overfitting to one. The experience replay stores old and new topologies; this means that if a satellite is lost, a new path has already been found, and when the satellite is returned, the old path is restored. The ε-greedy method also ensures continued exploration during training, preventing convergence to suboptimal routing policies and enabling adaptation to topology changes. The DQN agent itself is using four major action sets: the route path, the spectrum allocation, the power allocation, and the beam time allocation. These sets are broken up as follows:

route path $a \in \{path1,\ path2,\ \ldots,\ pathn\}$,

bandwidth $a \in \{low,\ medium,\ high\}$,
power $a \in \{low,\ medium,\ high\}$,
beam_time $a \in \{short,\ medium,\ long\}$

The action sets for spectrum allocation, power allocation, and beam time allocation are treated as discrete in this system, but in physical reality they are continuous, making this a very simplified model. The state of the agent is provided from the ground station; this is the satellite's current position and the position of the ground station, using standard ECEF coordinates. From this, it has the calculated distance north, east, and up (altitude), using the ECEF to locally levelled ENU equation, seen as Equation 7.

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} -\sin\lambda & \cos\lambda & 0 \\ -\sin\phi\cos\lambda & \sin\phi\sin\lambda & \cos\phi \\ \cos\phi\cos\lambda & \cos\phi\sin\lambda & \sin\phi \end{bmatrix} \begin{bmatrix} x_s - x_g \\ y_s - y_g \\ z_s - z_g \end{bmatrix} \quad (7)[15]$$

Where λ is the longitude, ϕ is the latitude, x_s, y_s, z_s are the satellite coordinates, and x_g, y_g, z_g are the ground coordinates. The other states are the elevation angle and visibility flags. This creates a state set of:

$$s \in \{sat\_x,\ sat\_y,\ sat\_z,\ gs\_x,\ gs\_y,\ gs\_z, relative\_distance, elevation\_angle, visibility\_flag\}$$

The environment provides these states along with the rewards, which will be based on typical qualities of satellite links, such as latency, doppler shifts, and fading. A possible reward system would look like the following:

$$reward\ +=\ (w1\ *\ throughput\ -\ w2\ *\ latency\ -\ w3\ *\ doppler\_penalty\ -\ w4\ *\ fading\_penalty\ -\ w5\ *\ power\_usage\ -\ w6\ *\ packet\_loss)$$

With the w referring to weights that allow control over what is wanted from the satellite. For example, if we want to focus on lower power, then $w5$ would be raised, but if we do not care about the power usage and want to focus our attention on latency, we would lower $w5$ and raise $w2$; however, each metric should be normalized to prevent dominance of any single term.

Overall, this agent learns a mapping from satellite-ground geometry to optimal routing and resource allocation decisions that maximize link performance under varying network conditions.

To further the discussion, this agent is not just a single agent for all satellites; it would instead be using a method called Multi-Agent Reinforcement Learning (MARL), so one agent is one satellite. The total reward can either be a network-wide performance for cooperation or a local link, which is easier to train but not as cooperative. This agent is better suited for use with

the network-wide performance to allow the possibility of prediction, since each agent has a constantly changing environment. The cooperative reward allows agents to learn of many possible outcomes, such as possible congestion seen by a previous satellite, or it will know if another satellite will be covering a region soon. Using a technique called Centralized Training, Decentralized Execution (CTDE) means that the agent has access to the global state as well as other agents' actions, but during execution, it uses only the local observations. In summary, the proposed MARL formulation with CTDE provides a distributed yet coordinated control paradigm for VHTS systems. The combination of centralized training and decentralized execution enables agents to learn globally optimal policies while maintaining local autonomy, thereby improving scalability and robustness in highly dynamic satellite network environments.

VIII. References

[1] Y. An, B. Di, H. Zhang, and L. Song, *"Evolution of high throughput satellite systems: Vision, requirements, and key technologies,"* IEEE Wireless Communications, vol. 27, no. 5, pp. 22–28, 2020

[2] S. K. Sharma, S. Chatzinotas, and B. Ottersten, "*Multibeam high through-put satellite: Hardware foundation, resource allocation, and precoding,*" IEEE Communications Magazine, vol. 59, no. 1, pp. 92–98, 2021.

[3] J. M. Maral and M. Bousquet, "*High throughput satellites,*" in Satellite Communications. IntechOpen, 2018, accessed: 2026-04-05. [Online]. Available: https://www.intechopen.com/chapters/60521

[4] SEA-MAN.org, *"High throughput satellites (hts),*" 2024, accessed: 2026-04-05. [Online]. Available: https://sea-man.org/hts-satellites.html

[5] J. Zhou, Y. Wan, Y. Li, and J. Wang, "*A deep reinforcement learning based dynamic resource allocation approach in satellite systems,*" IEEE Access, 2024. [Online]. Available: https://ieeexplore.ieee.org/document/11083701

[6] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction.*

[7] I. B. Levitan and L. K. Kaczmarek, *The Neuron: Cell and Molecular Biology*, 4th ed. New York, NY, USA: Oxford University Press, 2015.

[8] P. Poupart, "Lecture 3b: Annotated Slides," *CS885 Reinforcement Learning*, University of Waterloo, Waterloo, ON, Canada, 2025. [Online]. Available: https://cs.uwaterloo.ca/~ppoupart/teaching/cs885-winter25/slides/cs885-lecture3b-annotated.pdf

[9] Luong, N. C., Hoang, D. T., Gong, S., Niyato, D., Wang, P., Liang, Y.-C., & Kim, D. I. (2019). *Applications of deep reinforcement learning in communications and networking: A survey.* IEEE Communications Surveys & Tutorials, 21(4), 3133–3174. https://arxiv.org/abs/1810.07862

[10] Petrou, M., Tauran, B., Chasserat, L., Rebhi, M., & Pradas, D. (2025). *Improving user experience in hybrid SATCOM with deep Q-learning.* In Proceedings of IFIP Networking 2025. https://networking.ifip.org/2025/images/Net25_papers/1571143930.pdf

[11] Xie, X., Fan, K., Deng, W., Pappas, N., & Zhang, Q. (2025). *Multi-satellite beam hopping and power allocation using deep reinforcement learning*. arXiv.

https://arxiv.org/abs/2501.02309

[12] A. Ahmad, L. Du, and W. Zhang, "*Fast and practical Strassen's matrix multiplication using FPGAs,*" arXiv preprint arXiv:2406.02088, 2024.

[13] E. Cianca, T. Rossi, A. Yahalom, Y. Pinhasi, J. Farserotu, and C. Sacchi, *"EHF for satellite communications: The new broadband frontier,"* Proceedings of the IEEE, vol. 99, no. 11, pp. 1858–1881, 2011, doi: 10.1109/JPROC.2011.2158765.

[14] A. Kumar, N. Gaur, S. Chakravarthy, and A. Nanthaamornphong, "*Enhancing satellite networks with deep reinforcement learning: A focus on IoT connectivity and dynamic resource management,*" Results in Optics, vol. 18, art. no. 100765, 2025, doi: 10.1016/j.rio.2024.100765.

[15] J. S. Wight, "Radar and Navigation," ELEC 4600: Radar and Navigation, Carleton University, Ottawa, ON, Canada, 2025