Toward the Development of a Multi-Agent Cognitive Networking System for the Lunar Environment

Rachel Dudukovich, Dylan Gormley, Shilpa Kancharla, Katherine Wagner, Robert Short, David Brooks, Jason Fantl, Shruti Janardhanan, and Alexander Fung

***Abstract*—This paper details the development of a multi-agent cognitive system intended to optimize networking performance in the lunar environment. One concept of the future of lunar communication, LunaNet, outlines a complex network of networks. Challenges such as scalability, interoperability, and reliability must first be addressed to successfully fulfill this vision. Machine intelligence can greatly reduce the reliance on human operators and enable efficient operations for tasks such as scheduling and network management. Machine learning, artificial intelligence, and other automated decision-making techniques can be used to allow network nodes to intelligently sense and adapt to changes in the environment such as link disruptions, new nodes joining the network, and support for a diverse range of protocols. Cognitive networking seeks to evolve these technologies into an autonomous system with improved science data return, reliability, and scalability. In this paper, we study four main areas as a means to further develop cognitive networking capabilities: networking protocol development, analysis of wireless data for modeling and simulation, development of algorithms for a multi-agent system, and spectrum sensing technology.**

***Index Terms*—Cognitive networking, cognitive radio, delay-tolerant networking, multi-agent reinforcement learning, spectrum sensing, cross-layer optimization, link quality.**

I. Introduction

FUTURE plans for the lunar communication architecture detail a wide variety of missions. Within the time frame of 2018-2028, over 40 lunar missions are planned among multiple space agencies involving communications between lunar orbiters, surface mobile vehicles, surface stationary assets, and Earth ground stations [1]. Space inter-networking will be a key technology to develop, which must be capable of supporting lunar-Earth and lunar surface-lunar orbit links. The wide level of diversity among nodes makes inter-operability a challenge within multiple levels of the network stack. In addition, the network may be highly dynamic, requiring scalability and expandability. It is for these reasons that reliance on human operators and predefined communication schedules will become increasingly impractical. Cognitive networking seeks to develop algorithms and protocols that will sense, decide and act autonomously according to changes in the network. This paper outlines the development of a cognitive networking system prototype to address the challenges of the future lunar network.

The work discussed in this paper encompasses several key areas of the cognitive system including system architecture development, delay-tolerant networking, machine learning/artificial intelligence, and spectrum sensing. The networking concept relies upon the framework of delay-tolerant networking (DTN) to mitigate disruptions and to provide a common layer among nodes. This article builds upon a conference paper [2] of the same title. However, we expand our previous work in several key ways. We incorporate physical layer sensing capabilities to enhance the reliability of the system with hardware-level knowledge of the environment. In addition, we give an extensive discussion on spiking neural networks and multi-agent reinforcement algorithm development.

A. Lunar Scenario

National Aeronautics and Space Administration's (NASA's) LunaNet architecture outlines the network infrastructure that will support future missions to the lunar surface [3]. The lunar network will consist of a wide variety of nodes including mobile and stationary surface assets, orbiters, relays, and earth ground stations. Each LunaNet node will support three standard services: networking services, position, navigation, and timing (PNT) services, and science utilization services. Networking services will create an end-to-end path for data through a disconnected, time-varying topology, with many operations transparent to the user. Cognitive networking addresses many of the challenges presented in the LunaNet architecture; specifically unlimited scalability, interoperability, and enhanced reliably in a highly mobile, intermittently connected environment.

Of particular interest is the addition of small satellites under 500kg (smallsats) to the lunar architecture. Smallsats

Manuscript received January 14, 2022; revised March 8, 2022; accepted March 10, 2022. Date of publication March 30, 2022; date of current version June 10, 2022. The work was supported in part by the Cognitive Communications project funded by the NASA Space Communications and Navigation program. *(Corresponding author: Rachel Dudukovich.)*

Rachel Dudukovich, Dylan Gormley, and Robert Short are with the Communication and Intelligent Systems Division, NASA Glenn Research Center, Cleveland, OH 44135 USA (e-mail: rachel.m.dudukovich@nasa.gov; dylan.j.gormley@nasa.gov; robert.s.short@nasa.gov).

Shilpa Kancharla, Katherine Wagner, Jason Fantl, Shruti Janardhanan, and Alexander Fung were with the SCaN Internship Program, NASA Glenn Research Center, Cleveland, OH 44135 USA (e-mail: skancha@ncsu.edu; kwagne27@uic.edu; jfantl@uwyo.edu; sjanardhanan@umass.edu; alexfung@berkeley.edu).

David Brooks is with SAIC, Cleveland, OH 44135 USA (e-mail: dave.e.brooks@nasa.gov).

Digital Object Identifier 10.1109/JRFID.2022.3162952

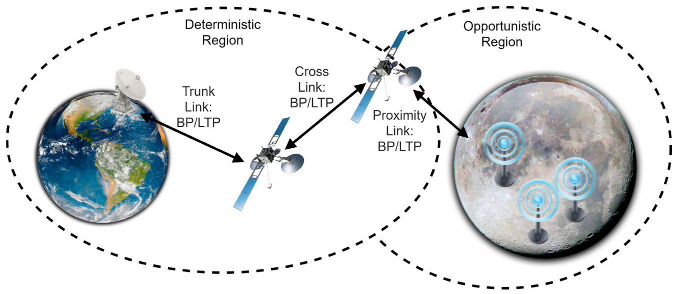

Fig. 1. Simple lunar architecture.

may provide critical capabilities within the LunaNet architecture including: acting as infrastructure elements providing networking and pointing, navigation, and tracking (PNT) services, as well as supporting science missions by carrying smaller instruments [4]. Instead of traditional network architectures with a few large spacecraft, these smallsats may operate as collaborative agents in the future, creating new types of constellations, formations, and clusters. This enables highly flexible multi-hop network systems.

The advent of software-defined radios (SDR) has allowed for even more flexibility and control over the communication system. SDRs will allow the radio to switch between NASA and commercial services by supporting multiple protocols. Links may be bi-directional or uni-directional and support low-rate command channels as well as high-rate data channels.

In this scenario, we envision two classes of networks as shown in Fig. 1. The first is a Mobile Ad hoc Network (MANET) on the lunar surface consisting of both robotic nodes mobile nodes and stationary nodes. Network disruptions may occur as vehicles enter lunar craters or move out of the range of other nodes. Delay/Disruption Tolerant Networking (DTN) protocols can be used to mitigate these disruptions and allow nodes to collect data for an extended period before needing to offload data to a lunar orbiter. This type of network may benefit from opportunistic styles of routing, in which a known network topology may be difficult to obtain.

The second type of network consists of the lunar orbiters, relays, and Earth ground stations. The connections in this type of network are well-defined, with known contact times and the number of participating nodes. This network will also benefit from the use of Bundle Protocol and Licklider Transmission Protocol (LTP) to account for periods of disconnection. However, this type of network is deterministic and is quite suited to Contact Graph Routing (CGR) [5].

To this end, we have developed an orbital scenario in the Satellite Orbit Analysis Program (SOAP) [6] to facilitate future testing. The scenario uses ephemera data from the Deep Space Gateway Near Rectilinear Halo Orbit [7] along with orbital information describing an "ideal" lunar communications relay system provided by the Inter-agency Operations Advisory Group [1]. The Gateway and relay satellites provide the predictable trunk line connection to Earth. From there, science missions in orbit and on the lunar surface occupy the opportunistic region closer to the moon. Fig. 2 depicts a part of this simulated scenario.

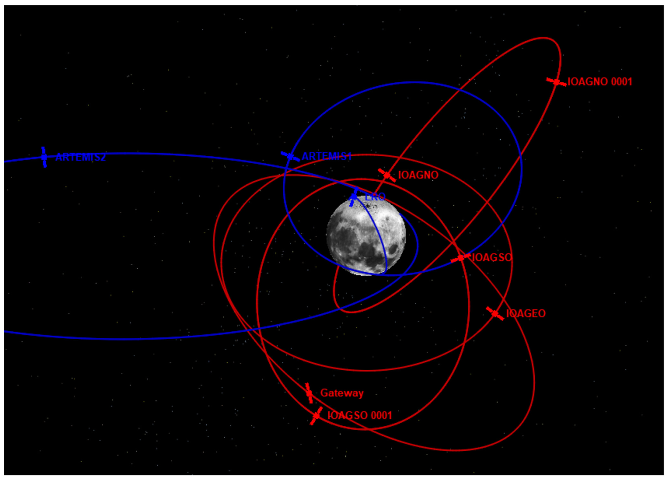

Fig. 2. Lunar scenario in SOAP.

II. Related Works

A wide breadth of technologies is needed to actualize a cognitive networking system. Considerations range from the overall network architecture and the role of network nodes, the network and application layer protocols, software algorithms for higher-level decision making, and physical layer sensing and control. The next subsections provide elementary background on each of these technologies.

A. Network Architectures

Cognitive network architectures have been an open research topic for NASA's Space Communication and Navigation (SCaN) program for several years. Initial work was developed as part of the SCaN Testbed project [8]. Early work explored the concept of distributed versus centralized decision making, cross-layer messaging, and basic elements of node discovery [9]. The *Cogent* architecture outlines several key aspects that are further developed in this work. The concept of cross-layer messaging between cognitive radio and cognitive network layer is introduced. *Cogent* uses small discovery messages to establish the presence of connectivity across a pair of network interfaces. The messages may also include metadata regarding link characteristics, such as latency and throughput, that are measured and provided as input to a cognitive engine. The *Cogent* concept may act as a protocol gateway that supports the use of Internet Protocol (IP), DTN, Consultative Committee for Space Data Systems (CCSDS), and commercial network stacks.

B. Low Size, Weight and Power Platforms

The technologies discussed throughout this paper are focused on enabling cognitive communications on low size, weight, and power (SWaP) platforms such as small satellites (smallsats) and rovers. There are a number of lunar science missions with plans to utilize smallsats to enable such tasks as studying water, ice, and minerals on the Moon [10], [11], [12], [13]. In addition to science missions, smallsats may plan a role in the future lunar communication infrastructure [4]. Lunar rovers have been used for science missions over the

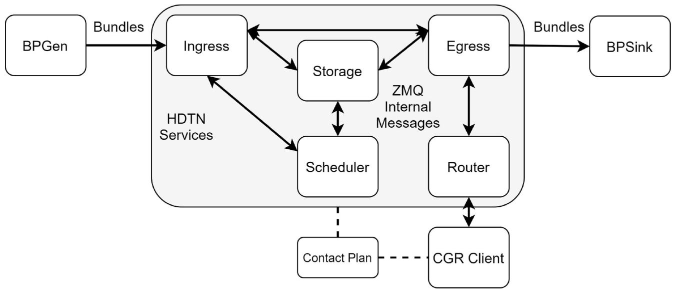

Fig. 3. High-level HDTN architecture.

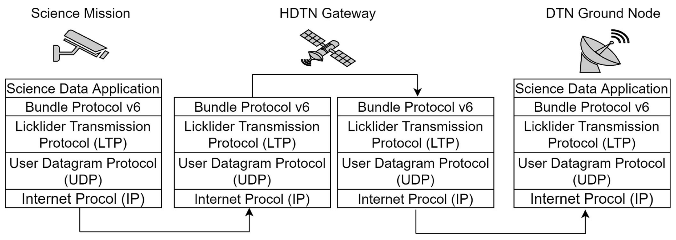

Fig. 4. Prototype network stack.

last several decades [14] for surface exploration. These low SWaP vehicles have enabled significant advances in science, in part due to the small, efficient, and more agile nature of such missions. Future lunar missions may benefit from mobile ad hoc networks comprised of these smaller science missions and infrastructure nodes. This type of use-case is a focus of many cognitive networking techniques [15] in which the network is increasingly dynamic. The technologies discussed in the next subsections including: delay tolerant networking, spectrum sensing, and neuromorphic computing must account for low SWaP considerations required by such platforms.

C. Delay Tolerant Networking

The LunaNet architecture specifies DTN Bundle Protocol (BP) as the primary internetworking protocol. The BP network overlay creates a common layer that is consistent across multiple heterogeneous network regions or subnets. Many works enumerate the drawbacks of applying strictly IP-based (Transmission Control Protocol and User Datagram Protocol) terrestrial protocols in environments with long propagation delays and intermittent connectivity [16].

The High-rate Delay-Tolerant Networking (HDTN) project at NASA Glenn Research Center has been developing a modular implementation of a DTN bundle agent [17], [18]. The HDTN software has been released under the NASA Open Source Agreement and is available on GitHub [19]. Previous work has been focused on the development of store-and-forward, UDP, TCP, STCP, and LTP convergence layers. Fig. 3 shows a high-level diagram of the main software components.

The network stack implemented in the prototype system is shown in Fig. 4. Licklider Transmission Protocol in addition to Bundle Protocol is featured in the lunar architecture [1] to support long-distance links with the reliable transfer. Links from the moon directly to Earth may require this capability. Wireless LAN nodes on the surface may function within their own subnetwork, which will transmit data to a lunar relay or lunar gateway as shown in Fig. 1.

D. Opportunistic Routing in Delay Tolerant Networks

There is a significant body of work discussing the advantages and disadvantages of deterministic versus opportunistic styles of routing in delay tolerant networks. Delay tolerant networking has been applied to several different types of use-cases. The first, which is closely related to the work in this paper, is space inter-networking. Routing for this type of network tends to focus on the periodicity of orbiting nodes and the fact that space communication, for NASA in particular, is generally scheduled well in advance. For these reasons, simple static routing or Contact Graph Routing is often deemed sufficient. The other main area of application for DTN is mobile ad hoc networks in challenging environments such as rural areas, recent disaster sites, and underwater networks. In this case, the network is much less structured and predictable. Many opportunistic routing methods use some form of calculating the likelihood of contact between pairs of nodes in the network. Our previous work focused extensively on applying algorithms based on machine learning and artificial intelligence to these types of scenarios [18], [20], [21], [22], [23].

Opportunistic Contact Graph Routing (OCGR) assigns a confidence level to each contact in the contact plan [24]. OCGR uses three main variables to control its behavior. The base confidence is used with the contact history to estimate the likelihood of a predicted contact between any two nodes. The minimum confidence improvement γ is a threshold in bundle delivery that a path needs to demonstrate to be considered a valid candidate. The third variable η controls the number of times a bundle will potentially be replicated to ensure delivery. To determine the overall confidence level, OCGR maintains two tables of contacts: the contact plan of predetermined contacts, and the contact history based on discovered nodes.

The initial steps of the cross-layer optimization discussed in this paper build upon the OCGR concept of using CGR-like shortest path-finding mechanisms, while adding a weight variable, updated through reinforcement learning methods, which controls path selection based on historical and predicted outcomes. Section II-F discusses the technologies necessary to incorporate physical layer measurements to generate optimize opportunistic routing decisions.

E. Multi-Agent Decision-Making

Effective CGR implementations have been explored using a variety of classical and machine learning algorithms with effective success [25]. However, these well-tested approaches necessarily resolve decisions at the level of the immediate contact and potentially waste processing resources to re-appraise routing with changes in network state. These approaches also face realistic issues in accommodating partial access to and interruption of node and network state information. Hybrid scenarios as described previously may benefit from occasional opportunistic exploitation of resource availability [26], [27], [28].

A more comprehensive model of the CGR problem in space networks captures the context as a multi-agent system. Multi-agent techniques are especially adept at optimizing decision-making in complex, distributed environments [29]. For the CGR problem in particular a multi-agent framework acknowledges node inter-connectivity and the progressive evolution of network state shaped by the interacting decisions of the node-agents [30].

In this scenario, each contact acts as a semi-independent collaborating decision-maker in a shared setting to produce a set of routing decisions that inform the decisions made by subsequent points of contact [29]. Using a reinforcement learning approach, the multiple agent nodes linked in the network are exposed to a series of training scenarios and learn to select progressively optimized routing schemes in response to a reward applied to the decisions made every time-step [31]. The objective is a networked system that makes more efficient decisions by acknowledging the action of each node [32].

Of particular interest for comparison are two chief MARL algorithms: Deep Q-learning and Advantage Actor-Critic. Both algorithms have been widely explored in multi-agent learning contexts and have shown considerable promise in collaborative and distributed multi-agent settings, including in networking problems [33], [34].

In Deep Q-learning [35], a Q-value function is estimated in each time-step to refine the rational value assigned to each action available to the agent. Through training, this value function shapes an optimal process of decision making by each agent node that is sensitive to the decisions made by the other agent nodes [31], [36].

Advantage Actor-Critic (A2C) [37] in contrast uses a centralized value function to "critique" and subsequently shape the particular utility assessment with which each individual node actor makes a decision. Replacing the use of the Q-function, the Advantage function computes a relative value of each action as informed by the cumulative choices of all node-actors compared to an average action choice. Since the critic-assessed advantage function is updated more frequently than that of the actors, on each time-step the actor value function is purported to improve more rapidly with critic influence than without [32], [38].

F. Streaming-SCA Spectrum Sensor

A cognitive agent must be able to sense its environment to assess possible decisions that may earn it the greatest reward. While the previous sections focused on the network layer and above, the communication system will only be fully optimized if physical layer information is conveyed throughout the system since the network and higher layers ultimately depend on the quality of the lower layers. To this end, we envision spectrum sensing technology will play a key role in enabling cognitive networking capabilities. There are many spectrum sensing detection methods available. In this subsection, we will review three common methods used to detect interference.

1) Matched Filter Detector: This technique is optimal when we know the signal of interest, but not when it is active/inactive. To increase the robustness of our cognitive network, we wish to identify characteristics of *unknown* signals. Therefore, this technique is not suitable for our network's blind detection requirement.

2) Energy Detector: This technique is common as a simple method to blindly detect RF features of unknown signals. However, this system suffers from a high rate of false alarms in environments with changing noise levels, such as space. If our system misses signal detection, we could interfere with a neighboring transmission. On the other hand, if our system detects a signal that is not there, we have missed an opportunity for transmission. Therefore, this technique is not suitable for our network's architecture.

3) Cyclostationary Feature Detector: This technique is desirable when signal detection must be performed blindly and with high accuracy. Thus, it is typically considered one of the best choices for detection. However, this technique is extremely computationally expensive. As such, it is traditionally limited to use with terrestrial receivers. Therefore, this technique is not suitable for our network's low size, weight, and power (SWaP) smallsats [39].

However, there has been a recent development of a low-SWaP Cyclostationary Feature Detector application known as the *Streaming-SCA Spectrum Sensor* [39], [40] (henceforth referred to as simply the *sensor*). This sensor is compatible with smallsats, can identify RF signals blindly, and maintains high levels of accuracy even through low SNR values and co-channel interference.

G. Introduction to Neuromorphic Computing

Formally, artificial intelligence program is defined as a software system that can sense, reason, act, and adapt [41]. The first generation of artificial intelligence was a rules-based structure that emulated classical logic to draw reasoned conclusions within some specific problem domain. The second and current generation of artificial intelligence focuses on sensing and perception, as exemplified by deep learning networks. As the field progresses, the focus turns to ensure artificial intelligence can be on par with human cognition, which includes tasks such as interpretation and autonomous adaptation. This transition is imperative to overcoming the brittleness of many such algorithms. This brittleness stems from operating in a deterministic environment that does not necessarily have access to situational context and the true stochastic nature of scenarios.

Neuromorphic computing involves the use of circuits that emulate biological structures in the nervous system. Such structures offer possible algorithms that can deal with the ambiguity found in the everyday world. The next generation of artificial intelligence should strive to account for novel situations to automate human-like activity. The key challenges of neuromorphic computing work to address matching a human's adaptability and ability to respond to environmental stimuli [42]. Our current work focuses on spiking neural networks (SNNs), a type of non-traditional deep learning model that can be trained on-chip using a new generation of neuromorphic processors [43].

While artificial neural networks (ANNs) perform reasonably on traditional CPU and GPU architectures, spiking neural networks will demonstrate improved performance on specialized neuromorphic hardware. In addition to performance benefits, current research indicates neuromorphic processors may allow for complex computing on low power platforms [44]. For this reason, neuromorphic processors may be a key component to enabling artificial intelligence in the space environment.

III. Spiking Neural Network Development

In this paper, we consider the benefits and relationship between two major subsets within the field of artificial intelligence and machine learning: artificial neural networks and reinforcement learning. Both approaches may provide key roles towards learning optimal network policies. Artificial neural networks can model complex functions based on observed data. Current work in the field of neuromorphic computing may reduce the SWaP requirements of traditional processors and GPUs; thereby making techniques such as spiking neural networks quite feasible for the space environment. Reinforcement learning will provide the capability to modify network models and policies online, as a reaction to current changes in the network.

This section focuses on evaluating publicly available data sets, developing concepts for a SNN predictive model, and simulation results. An offline learning approach was used to perform data exploration, feature engineering, as well development of the initial SNN model. Future work will seek to integrate this model into the overall cognitive system and incorporate online learning capabilities.

A. Data Set Analysis

As a preliminary step towards an ANN or SNN model, algorithm development, and implementation, we sought publicly available data sets related to delay-tolerant and mobile ad hoc networks for preliminary analysis. Eventually, three main data sets became the focus of our work: the ORBIT testbed data set [45], the DieselNet data [46], and the Barcelona Neural Networking Center (BNNC) Graph Neural Networking Challenge 2021 Data Set [47].

1) ORBIT Data Set: Our initial data set consists of measurements from the Rutgers Open-Access Research Testbed for Next-Generation Wireless Networks (ORBIT) [45]. This data set includes the received signal strength indicator (RSSI) for each correctly received frame at the receiver node when various levels of noise are injected on the ORBIT testbed. The Rutgers ORBIT testbed data set is particularly of interest since the Cognitive Communications project at NASA Glenn is building an RF testbed similar to the ORBIT concept. The ORBIT testbed data can give insight into metrics and data set formats useful for machine learning.

The features used in this analysis include the following:

- *Received:* whether the signal was received or not (Boolean value)
- *Error:* indicates if an error has occurred while capturing the RSSI (Boolean value)
- *Noise:* the amount of noise injected
- *t_x:* x-coordinate of the grid node that was configured as the transmitter (integer value)
- *t_y:* y-coordinate of the grid node that was configured as the transmitter (integer value)
- *r_x:* x-coordinate of the receiver node (integer value)
- *r_y:* y-coordinate of the receiver node (integer value)

The preprocessed data set resulted in 1,218,000 data points. Our initial problem formulation was to predict the RSSI value using several regression techniques. The methods used were: multiple linear regression, ridge regression (L2-norm), LASSO regression (L1-norm), random forest regression, Bayesian ridge regression, and finally XGBoost regression. To assess the performance of the regression models, the root mean squared error (RMSE) was being used. The RMSE measures the average magnitude of the error. Since the errors are squared before they are averaged, the RMSE gives a relatively high weight to large errors. To understand if large errors are present, the RMSE is particularly useful. A training, validation, and test split of the original dataset was created at 75%, 12.5%, and 12.5% respectively.

Lastly, the optimal hyperparameters used for XGBoost regression were found using AWS SageMaker. These hyperparameters were used to inform the model created by hand as well. This approach of both "hand-done" and automated tuning tended to yield the best result. Ultimately, we concluded that additional higher layer metrics such as packet loss and per-packet delay combined with physical layer metrics, such as the RSSI value would be more relevant to a cognitive networking system. For this reason, we continued to explore additional data sets.

2) DieselNet Data Set: While the ORBIT data set includes interesting characteristics such as varying noise levels injected into the system, the data set is based on stationary nodes that do not truly emulate a DTN environment. For this reason, we evaluated the DieselNet data set [46]. DieselNet is a network of 35 buses on the campus of the University of Massachusetts, Amherst, which traveled planned routes every day throughout 5 months in the Spring 2006 semester. The buses were equipped with radio transmitters, and the trace set includes data for each one-way connection when the buses came into close enough proximity to transfer data. The data set provided 60,000 one-way contacts.

DieselNet approximates a terrestrial DTN environment well due to several factors. The link quality in DieselNet can be measured by throughput or the amount of data transferred over a connection per unit time. For our lunar scenario, bus mobility roughly simulates the movement of vehicles on the lunar surface. The regularity of bus routes could be envisioned as the movements of lunar rovers performing periodic maintenance activities. However, there are also weaknesses with this data set. Data collection was imperfect: the GPS coordinates were the same for both buses for each contact, the reported time duration was negative for a small fraction of the contacts, and some were missing data fields. After filtering these out, about 50,000 of the initial 60,000 contacts remained. However, many of these were "repeat" contacts – the same two buses phased in and out of connection up to roughly ten times. We ultimately concluded the DieselNet scenario and data set did not resemble the space environment closely enough to warrant further investigation of the data.

3) BNNC Graph Neural Networking Challenge Data Set: The data set we found most useful was provided by the Barcelona Neural Networking Center (BNNC) for their Graph Neural Networking (GNN) Challenge 2021, in which the aim was to create a scalable network digital twin using graph neural networks. By providing a snapshot of a network state, the goal was to create a scalable network digital twin based on neural networks that accurately estimate various performance metrics of a network [47]. Our work adapts the BNNC data set for a spiking neural network application. Each sample simulates a network scenario which comprises of three different aspects:

- Network topology
- Routing configuration
- Source-destination traffic matrix

The scenarios are labeled with network performance metrics defined by the simulator, including per-source-destination performance measurements (mean per-packet delay, jitter, and loss), and port statistics (e.g., queue utilization, average packet loss). The focus of the GNN Challenge was to predict the mean per-packet delay on each source-destination path using GNNs and the provided data set.

The BNNC Graph Neural Networking Challenge Data Set is currently the most complete and relevant data set for the project that we have found. This data set was meant to be used with Graph Neural Networks rather than traditional ANNs or SNNs. During our preprocessing of the data set, we came upon challenges for representing graph-structured data in more traditional formats (tabular data). Since networks can be represented as graphs, our future work will dive more deeply into the data representation as well as GNNs. Existing work shows GNNs may be useful for network modeling [48].

The training dataset provided by BNNC contains samples simulated in topologies of 25 to 50 nodes, including two different network topologies for each topology size. The validation dataset provided by BNNC contains samples on larger network topologies up to 300 nodes. The test dataset provided also has a similar layout to the validation dataset.

B. Spiking Neural Network Approach

Spiking neural networks are different from traditional artificial neural networks as they incorporate time as a dependency within their computations. Neurons are still considered the basic processing elements. However, at some instance in time, one or more neurons might send out impulses, or a spike, to neighboring neurons through connections known as synapses. The travel time is considered zero for this. Neurons also exist in a local state with rules governing their actions and timing of spike generation. This network is a dynamic system where individual neurons interact through spikes. One of the main differences between modern deep learning and how the brain functions are that the brain encodes information in spikes rather than continuous activations. SNNs are on the spectrum of ANNs that closely mimic natural, neurobiological neural networks. Key features of the SNN structure include information about the neuronal and synaptic state alongside time. Another highlight of SNNs is that they perform favorably on neuromorphic hardware with properties such as low power consumption, fast inference, and event-driven information processing [49].

After finding a suitable data set as discussed in the previous section, the next step of our work was to begin to understand how to process this data using an SNN. `snnTorch` is a Python package for performing gradient-based learning with SNNs [50]. It extends the capabilities of PyTorch, taking advantage of its GPU accelerated tensor computation and applying it to networks of spiking neurons. Predesigned spiking neuron models are integrated within the PyTorch framework and are treated as recurrent activation units. While all of the available models in `snnTorch` can run on CPU, they can be loaded onto CUDA and run on GPU.

1) Data Encoding: SNNs are made to exploit time-varying data. If the goal is to create an SNN, it makes sense to use spikes as inputs too. However, not all datasets are time-varying, such as the BNNC dataset described. It is quite common to use non-spikes as well. Two methods may be used to process more traditional datasets (such as the well-known MNIST image dataset) that do not vary with time. The first method is to simply pass the same data sample into the network at each time step. While this approach is quite straightforward, it does not fully exploit the temporal dynamics of SNNs. This approach was used for the BNNC dataset for the current set of experiments. However, it is worth discussing the second method, which is to convert the data into a spike train of a certain sequence length and input it into the network. While this method is more complicated, it gives insight into what data characteristics to potentially look for when collecting data in the future if the goal is to employ an SNN to process it. The module `snntorch.spikegen` (referring to spike generation) contains a series of functions that simplify the conversion of data into spikes. There are currently three ways to convert the data into spikes:

- *Rate coding:* uses input features to determine spiking frequency
- *Latency coding:* uses input features to determine spike timing
- *Delta modulation:* uses the temporal change of input features to generate spikes.

In rate coding, each normalized input feature $\mathbb{X}_{ij}$ is used as the probability an event (spike) occurs at any given time step, returning a rate-coded value $\mathbb{R}_{ij}$. This can be treated as a Bernoulli trial: $R_{ij} \sim B(n, p)$ where the number of trials is $n = 1$, and the probability of success (spiking) is $p = \mathbb{X}_{ij}$. Explicitly, the probability that a spike occurs is $P(R_{ij} = 1) = \mathbb{X}_{ij} = 1 - P(R_{ij} = 0)$. A Bernoulli event is an event for which the probability of occurrence is p and the probability of the event not occurring is $1 - p$. The event has two possible outcomes. A Bernoulli process is a sequence of Bernoulli trials. Among other conclusions that could be drawn, for n trials, the probability of n successes is p^n.

$$P(n) = \begin{cases} 1 - p & \text{for } n = 0 \\ p & \text{for } n = 1. \end{cases}$$

In rate coding, input features are used to parameterize a binomial distribution, which is then sampled from to determine whether or not a spike occurs. If the input falls outside of [0, 1], this no longer represents a probability. Such cases are automatically clipped to ensure the feature represents a probability.

In latency coding, each feature corresponds to a single spike. The intensity of the feature determines how fast the spike occurs. Options for linear or logarithmic firing times are available. Temporal codes capture information about the precise firing time of neurons. A signal spike carries much more meaning than in rate codes which rely on firing frequency. Features closer to 1, will fire earlier and features closer to 0 will fire later. Larger input means a fast spike, while a small input means a late spike. There are two arguments to pass in when performing latency coding. The first argument is τ. The higher the τ value is, the slower the firing will be. The second argument is the threshold, which represents the membrane potential before firing. One major advantage of latency coding over rate coding is the increased sparsity of spikes. If neurons are constrained to firing a maximum of once over time, then this promotes low-power operation.

Delta modulation is based on event-driven spiking, as biology is event-driven and neurons rely on change. It takes the difference between each subsequent feature across all time steps. By default, if the difference is both positive and greater than the threshold, a spike is generated.

2) Spiking Neuron Models: There is a range of neuron models to consider. The spectrum ranges from the biophysically accurate Hodgkin-Huxley model to the simplified artificial neuron that is rife in deep learning. In between both the Hodgkin-Huxley model and the artificial neuron is the leaky integrate-and-fire (LIF) neuron, model. It takes the sum of weighted inputs, much like an artificial neuron. Rather than passing it directly to an activation function, it will integrate the input over time with leakage, much like an RC circuit. If the integrated value exceeds the threshold, then the LIF neuron will emit a voltage spike. The LIF neuron abstracts away the shape and the profile of the output spike. It is simply treated as a discrete event. As a result, information is not stored within the spike, but rather the timing (or frequency) of spikes. Currently, `snnTorch` supports four types of LIF neurons:

- Lapicque's RC model
- Non-physical first order model (also known as leaky neuron model)
- Synaptic conductance-based neuron model
- Alpha neuron model

For the experiments in this project, the non-physical first-order model and synaptic conductance-based neuron model are used. The synaptic conductance-based LIF neuron model differs from the LIF neuron model as it takes into account the more realistic scenario of the decay of the synaptic current.

3) Simulation Approach: The BNNC data set was selected to develop the SNN predictive model, due to the extensive metrics found in the data set, as well as the data volume (46.6 GB worth of tabular data after preprocessing) and overall quality. The main target of focus is the average packet loss in this experiment. It is hypothesized that a deep learning regression model can be applied to predict any target of interest from the variables described for `linkUsage.txt`.

TABLE I
SNN TRAINING AND TEST SET SUMMARY

Set Sample	Feature Shape	Target Shape
Train (All Features)	$(2338834, 21)$	$(2338834, 1)$
Test (All Features)	$(662343, 21)$	$(662343, 1)$
Train (Feature Engineering)	$(2338834, 11)$	$(2338834, 1)$
Test (Feature Engineering)	$(662343, 11)$	$(662343, 1)$

The initial set of features selected were based on the data found in `simulationResults.txt` and `linkUsage.txt`. In addition to the target variables that were dropped, the Boolean feature of whether the link exists or not was dropped. In addition, if the link did not exist, there was a -1 in place for all the link usage measurements. These rows were dropped as well, as focusing on the input for which links exist would produce a more accurate regression model. This manual selection resulted in 21 input features, and therefore 21 nodes in the input layer. There are 1000 nodes in the hidden layer. The output layer will contain one node as this is a deep learning regression problem of predicting the value of the average packet loss based on the other features. The number of nodes in the hidden layer was inspired by the `snnTorch` tutorials given in the documentation, although this can also become a hyperparameter that can be further tuned for models in the future.

Principle Component Analysis (PCA) was used to further reduce the dimension of the data, resulting in a smaller data set consisting of 11 features. We consider both the initial data set (without feature engineering) and the reduced data set (with feature engineering).

Table I summarizes the train and test sets both with and without feature engineering. In order to be fed into the model, the data must be of type tensor. Tensors are dimensional data structures that can exist in dimensions ranging from 0 to n. For the data to be fed into the SNN, it must be of the format $[time \times batchsize \times feature\_dimensions]$. In addition, for the data to be run in a PyTorch deep learning model, the `DataLoader` object must be used to create and iterator that loads batches of data into the model. The goal is to create an input spike train to pass into the network. Arbitrarily, 25 time steps were chosen to simulate across the 21 input neurons, so the dimensionality is 25×21 in the case without feature selection, and 25×11 using the PCA reduced data set. Neural networks process data in minibatches, and the minibatch size chosen for this experiment was 128. Therefore, the dimensions as each input spike is passed through the SNN is $[25 \times 128 \times 21]$ for both the training and test sets without feature selection. With feature selection, the shape becomes $[25 \times 128 \times 11]$. Moreover, if there was extra data that did not meet the dimension standards originally set by the `DataLoader` object, that batch was dropped. This was done by setting the `DataLoader` object's parameter of `drop_last` to `True`.

4) Modeling Results: The results of the accuracy, as percentages, for the training and test data sets for the leaky SNN

TABLE II
ACCURACY RESULTS OF LEAKY AND SYNAPTIC MODELS (ALL FEATURES) (%)

	Leaky	Synaptic
Train	95.31	92.97
Test	89.84	89.84

TABLE III
ACCURACY RESULTS OF LEAKY AND SYNAPTIC MODELS (FEATURE ENGINEERING) (%)

	Leaky	Synaptic
Train	92.97	96.09
Test	91.41	89.06

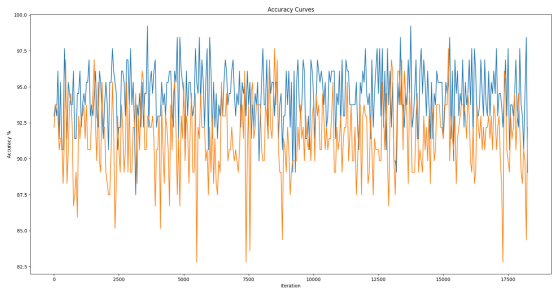

Fig. 5. Accuracies of training and test data for the leaky SNN model without feature engineering.

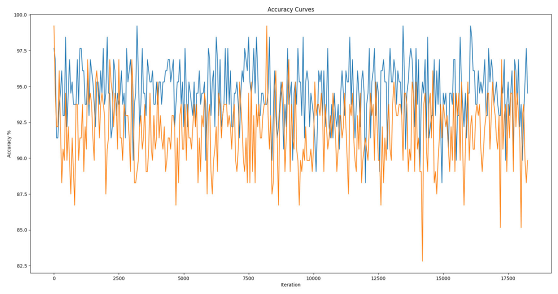

Fig. 6. Accuracies of training and test data for the synaptic SNN model without feature engineering.

and synaptic SNN are shown in Table II. These results were gathered over the course of only one epoch. After one epoch, the final accuracies of the leaky SNN and synaptic SNN are both 89.94% on the test data. In addition, the results when applying feature engineering to the model are also provided to attempt to determine if reducing the dimensionality of the data provides an improvement in loss and accuracy for both types of SNN models. The accuracy results when feature engineering is applied are shown in Table III. After one epoch as well, the final accuracies of the leaky SNN and synaptic SNN are both 91.41% and 89.06%, respectively, on the test data. The training and test accuracies of the leaky SNN model without feature engineering are shown in Figure 5. The training and test accuracies of the synaptic SNN model without feature selection are shown in Figure 6.

In the instance with the leaky SNN model without feature engineering, this model may contain some overfitting, as there is the greatest discrepancy between the training and test accuracy compared to all the other models. Interestingly, a similar discrepancy is present in the synaptic SNN model with feature selection in Table III. However, when running the leaky SNN model with feature selection, there isn't as great of a difference in accuracy between the test and training sets. It could be hypothesized from this that the leaky SNN model is less prone to overfitting, or may even be more suitable when there are fewer features present. However, the synaptic SNN model may be more suitable to scenarios when more features are present. It would be important to further test this hypothesis on various data sets for which feature selection can be performed.

IV. MULTI-AGENT REINFORCEMENT LEARNING APPROACH

While SNNs may prove to be a useful decision-making element, our current simulations have been conducted with offline learning using predefined data sets. In a dynamic network, decision-making will need to occur online so that the system will be able to react to changes in near real-time. For this reason, we investigate several multi-agent reinforcement learning approaches, with the thought that SNNs may be incorporated at some point as the function approximation mechanism as discussed in deep Q-Networks.

Our initial multi-agent reinforcement learning (MARL) approach abstracts many of the physical layer details away from the problem. While this may be an oversimplification, it allows for basic algorithmic concepts and simulations to be developed. The next section gives an overview of a basic MARL approach to cognitive routing.

This routing strategy takes an opportunistic and online approach. The network nodes detect neighboring communication nodes at the time of data bundle transmission and use one or several link quality assessment metrics, possibly in association with a node-pair connection history, to compare the set of prospective one-hop connection links and select the optimal link for transmission to the next node. Compared to CGR approaches, these ad hoc strategies are well-equipped to enhance the dynamic adaptability of space networks and address unexpected changes in network topology and failures at the level of the individual communication node. Opportunistic routing strategies also enable the prospect of discovery of network routing "short-cuts": routing paths overlooked or unplanned in contact graphs that optimize transmission metrics, potentially in conjunction with the use of other system resources [26], [27], [28].

These opportunistic approaches have been widely confronted using reinforcement learning algorithms able to confer these online benefits with minimal training and across network types. This learning framework reinforces optimal action selection in each decision making context by providing agents that take a certain action in response to an environmental observation with a numerical reward that reflects the value of that action for optimizing a target system metric. Agents are motivated to maximize these rewards, driving learning of optimal decision making through a variety of value-assigning

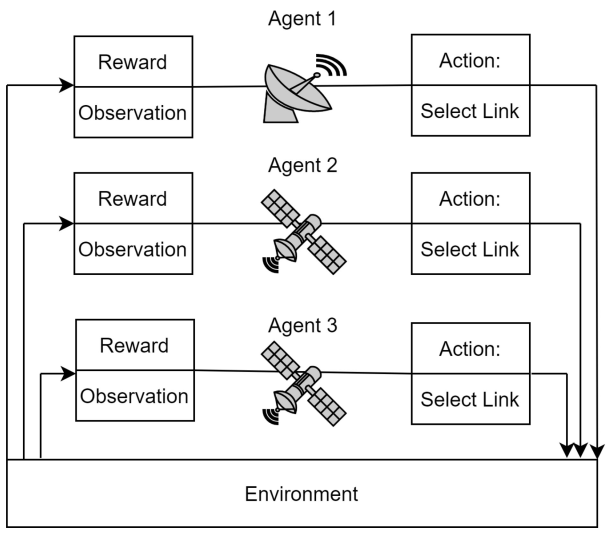

Fig. 7. Multi-agent reinforcement learning model.

functions [51]. An interpretation of these basic learning frameworks applied to network settings are depicted in Fig. 7. These network learning frameworks typically consider the agent to be the communication node preparing to pass a packet of data to its next hop and the action set as the space of possible transmission manipulations that may be applied to the current data bundle. These include selection of one of the links from the set of proximate neighbor nodes and, in DTN scenarios, the option to store the data bundle in the node agent's own buffer (delaying immediate transmission).

Agent-nodes' action choices are delivered to the processing unit that records and updates the state of the network environment, and then returns to the node-agent the observation-reward pair that reflects the value assigned to the change in network state and ultimately drives node-agent learning and improved selection of transmission links in subsequent time steps.

The reward and observation received by the agent varies across the literature for the dynamics of transmission in a particular implementation. Reward values feature both singular and composite interpretations of the value that the selected action conferred to optimizing network metrics. These have included the negated network response time [52], physical layer characteristics of the next hop (signal-to-interference-plus-noise ratio) [53], and an assortment of other measures of combined network node resource consumption. Observations capture the network state as a whole and often include information specific to individual nodes and node-pairs, including remaining queue capacity and estimated distances from possible next-hop recipients to the ultimate intended data destination.

The first generation of these applications took a single-agent approach to learning, with an assumption of autonomy, assigned to the focal node alone. Deriving from this assumption, single-agent frameworks shape node-agent decisions based on analysis of a generalized network environment, in which the actions of other network nodes are indistinguishably combined and treated as environmental noise. Focus on these frameworks in the first generation of route-learning approaches stemmed from both adequate performance through application of the routing algorithm and the presumption of a lack of (especially movement) control among non-focal network nodes in common, terrestrial mobile routing contexts [54].

These initial approaches opened the doors to the benefits of taking a learning approach to network routing. In practice, learning in these frameworks suffered from an inability to produce optimal decision-making in contexts where the focal agent reasoning and learning occurred in proximity to that of other similarly-reasoning agents - contexts where the decisions and learned-policy updates made by fellow agents influence the environment state and its evolution in unpredictable ways. These interactions degrade the previous stability of the environment, preventing the agent from relying on a predictable association between action and outcome [55]. The result is a combinatorial explosion of state space that outpaces the ability of processing resources to trace dynamic changes and a failure of learning to converge to the optimal policy.

The response to these issues was to isolate and consider directly the influence of the decision making of each agent on the networking environment. This perspective reframes the networking problem as a multi-agent problem wherein the intermingling impact of the many agents on the networking environment is considered explicitly.

To capture the influence of multiple agents on the environment, the current state is passed through observations explicitly including attributes of the multi-agent nature of the setting. Each agent computes its value function, shaped by its unique experience history and differential experience in a particular node neighborhood [29], [31], [56]. The result is a networked system that makes more efficient networking decisions by acknowledging the action of each node, considering the impact of joint actions, and providing to each agent enhanced system-wide awareness [55], [57], [58].

Our general approach to multi-agent learning sought to produce a system in which each network node worked as an independently reasoning, equivalently-equipped autonomous agent that learned with experience how to select data packet manipulation actions to optimize expected future rewards reflective of key network metrics. Following a large proportion of the literature, in this preliminary work, we focused on minimization of source-to-destination transmission latency as the optimization target for learning. A goal for extension is to expand this metric to a composite measure that more broadly reflects key measures of link quality. The particular application of learning within this objective was the agent nodes' selection of 1) whether to store the current data packet in its queue or pass it on to a new network node and 2) in the case of the latter, the particular node (in the form of node-identified link) to which to send the data.

1) Multi-Agent Simulation: We framed the dynamics of the simulated network environment in the model established by OpenAI Gym [59]. The model permits custom definitions of step-wise, dynamic, and partially-observable environments that support Markovian decision-making and align with the computation framework necessary for the implementation of the action-observation-reward cycle that characterizes reinforcement learning. In addition to its accessibility, the Gym model was also selected for its compatibility with Ray learning models. Ray [60] is an open source Python machine learning library focused on distributed learning. It provides an API for multi-agent environments as well as a number of well-known multi-agent models to train with. It also has the ability to train a number of models on a distributed system, which can improve training times.

We selected multi-agent-compatible Deep Q-learning [35] as a the first target for model development and testing. In traditional Deep Q-learning, with state-action pairs as input, a neural network estimates the output of a Q-value function that computes the total expected reward associated with a sequence of particular actions in a learning environment. During decision-making, processing of the current state input yields a value assigned to each possible action; the agent selects that highest-rated action to secure the greatest reward prospect, shaping optimal action sequences [33], [34].

In response to these shortcomings, the literature has effectively modified core Q-learning concepts - along with several branches - to better tune the approach to the complexity of multi-agent learning [55], [61]. One of the first and most basic variations [62] addressed the challenge of multiple agents by simply distributing the reasoning machinery across agents. This independent Q-learning provided each agent with its memory and set of neural networks to approximate an agent-specific Q-function and build a unique policy resolved to the specific and differential experience history of that agent. The approach led to a leap in route learning performance [63] and continues to be frequently used in network learning problems [57].

Our Deep Q-learning implementation followed this basic modification strategy, using Ray tools to provide each node-agent its Q-network, target network, and experience replay memory buffer to enable independent, equivalent, and parallel reasoning by each agent [62], [63]. Notably, the multi-agent capabilities of the independent Q approach in large and complex network contexts can be countered by slowed learning, often in conjunction with the computational costs of processing by the number of neural networks [56], [64]. We selected this approach despite these issues as we felt a Deep Q approach offered a straightforward and accessible implementation compatible with several learning tools. This basic implementation also offered a valuable foundation for comparison against the considerable literature background that has used a similar approach. In addition, this could be used to build a space network-specific baseline against which to compare enhanced models.

We implemented the initial learning model using the Ray framework with underlying RLlib libraries. We used built-in Ray tools to access a Deep Q-network using a PyTorch foundation with a modified configuration to include a target network and experience replay buffer that would be applied to each agent. We injected into this framework the Gym-based Python network simulator that encompassed a node agent module definition. Internal Ray configuration modules allowed us to specify a small set of two distributed roll-out workers assigned to each agent and a comparable, though independent, initial

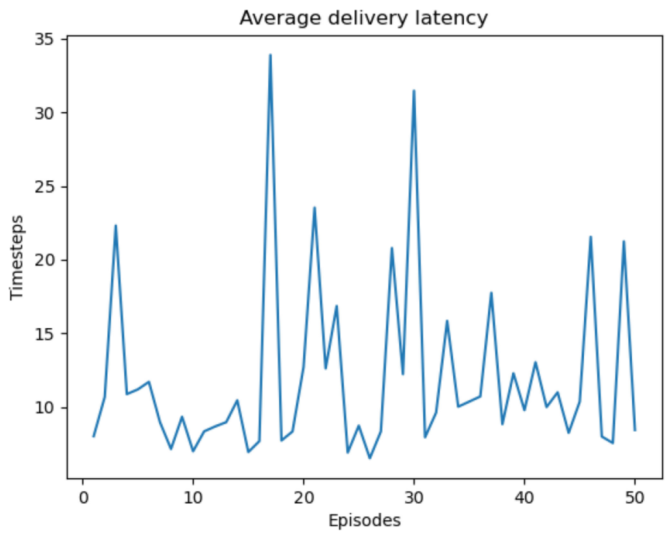

Fig. 8. Bundle delivery latency during training.

policy assigned to each agent. The result was a learning model that acted on each communication node in the network as an agent and attributed to each agent a unique set of Deep Q-networks for learning.

We applied this framework to a network comprising 50 similar agent-nodes. In the dynamics of the network simulation described above, routing decisions affecting the collection of randomly generated data packets were observed over 50 episodes of 1000 time steps each. Routing behavior was assessed at the level of the complete route path (i.e., source to ultimate destination). Focal metrics included the total per-packet transmission latency, the number of hops utilized in a complete route, and the proportion of packets successfully delivered to their ultimate destination within an episode. To evaluate the influence of packet traffic on performance, we additionally explored routing behavior under different packet generation constraints. We capped the number of packets generated during an episode to an arbitrary Minimum (500 packets), Moderate (1000), and High (1500) packet traffic conditions and recorded the described metrics resulting from each test run following learning. In support of future progress, here we present preliminary findings describing learning behavior using the Ray-enabled independent Deep Q-network approach in the simulated space network.

During training, per-packet delivery latency varied regularly between 8.1 and 18.7-time steps, with a typical packet reaching its destination within 11.4-time steps. The delivery latency profile produced during learning is displayed in Fig. 8. Despite the persistent variability, packets generated during the last quartile of episodes tended to be delivered to their destination 0.1-time steps more quickly than those generated during the first three quartiles. While this difference seems unlikely to have reached statistical significance, it is consistent with some degree of route learning by agent nodes. This potential learning was made more concrete during model testing under different conditions of network load, for the maximum number of generated packets. The interaction between load condition and packet delivery latency is demonstrated in Fig. 9.

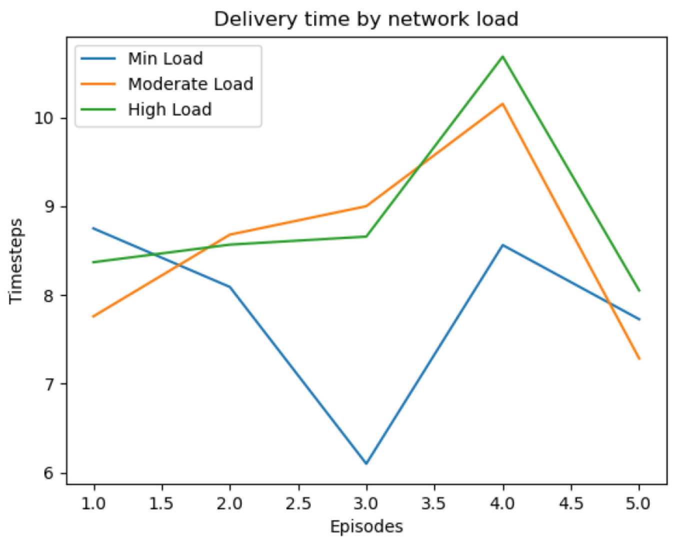

Fig. 9. Bundle delivery latency under varying network load.

V. RF-Feature-Enhanced Cognitive Network

Artificial intelligence and machine learning techniques, such as those found in Sections III and IV, address complex, long-term multi-agent patterns not easily recognizable to humans utilizing processing performed at a centralized node. However, these techniques currently lack the mechanics to adapt to real-time interference, risking mission failure. In this section, we discuss a method to enhance the robustness and decision-making of cognitive networks by utilizing the sensor discussed in Section II.

Unidirectional links have no form of feedback to verify if a transmission has been received. In this case, we first verify that the allocated spectrum is free as expected. If the sensor detects interference at the planned transmission frequency, the agent selects the next best contact from its contact plan. This is similar to the “listen to talk” method but differs in that it can tell the difference between noise and a modulated signal (user). This is important because noise is typically temporary (such as a solar flare), so we assume that we can safely transmit over this channel with some confidence link quality will improve. Transmitting over user interference is not desirable because the interference will likely be over a longer period.

The cognitive radios that will be used for bi-directional links will be using some form of an adaptive waveform, such as DVB-S2. One adaptive waveform of particular interest is described in [65]. This adaptive waveform utilizes a spectrum sensor. In this waveform, not only does the spectrum avoid/adapt to interference and noise, but the awareness is passed back to a mission operations center to incorporate into a database, enhancing the agent’s predictions.

A. System Architecture

While opportunistic algorithms may be a research topic that has been of interest for several decades, there is still a large gap in the technology required to implement this type of network in an actual spacecraft. For this reason, we feel the first step is to build upon the CGR concepts discussed in previous sections and focus on first achieving an intelligent system with both DTN and software-defined radio technology. This section will discuss the technology required for a single node prototype system. Each node is a software-defined radio platform and will have a data flow similar to Fig. 10. This is a general architecture that could be applied to a variety of intelligent nodes such as smallsats or mobile surface assets such as rovers.

In this initial architecture, a CGR-style contact plan is used to indicate a pre-planned contact schedule between a set of known nodes. In addition to data rates and contact time between nodes, the plan may also include service parameters such as frequency. To augment this plan, the spectrum sensor continuously processes received I/Q. It produces a list of where users are located and their bandwidth. The spectrum manager provides post-processing of the spectrum sensor by applying techniques such as time averaging and user time-to-live. The cross-layer engine cross-references the available contacts with the interferers. If the primary contact is unavailable due to interference, the list moves to a secondary contact, and so on.

Once the cross-layer engine has determined a contact to communicate with, it commands the link controller to configure the necessary waveform parameters such as modulation and coding, and the necessary RF parameters such as center frequency, gain, and sample rate. The waveform used is determined by the link controller. Of particular interest is the adaptation of the waveform’s modulation and coding using ACM. The RF Frontend’s receiver constantly receives I/Q samples centered at a given frequency and passes them to the spectrum sensor. The received samples are also received by the waveform, in parallel, when the waveform is enabled.

VI. Future Work

This section discusses observations from our current work and plans for potential improvements.

A. Algorithm Development

In the future, it may be useful to use a time-varying data set to fully exploit the temporal aspect of the SNN. There are three main characteristics for data when it comes to feeding in spiking input. The first characteristic is if the data contains ‘spikes’. Biological neurons communicate via spikes. Many computational models of neurons simplify this voltage burst into discrete, single-bit events: either a ‘1’ or ‘0’. This is far simpler to represent in hardware, such as a neuromorphic processor, than a high precision value. The second characteristic is sparsity. Neurons spend most of their time at rest, silencing most activations to zero at any given time. Not only are the sparse vectors/tensors (meaning they contain many zeros) cheap to store, but computations with them may be cheaper. We do not need to read many of the network parameters from memory. This means neuromorphic hardware can be extremely efficient. The third characteristic is static suppression or event-driven processing. Information is processed only when new information is present in the process. Conventional signal processing requires all channels or pixels (when using an image or video data) to adhere to a global sampling/shutter rate, which slows down how frequently sensing can take place.

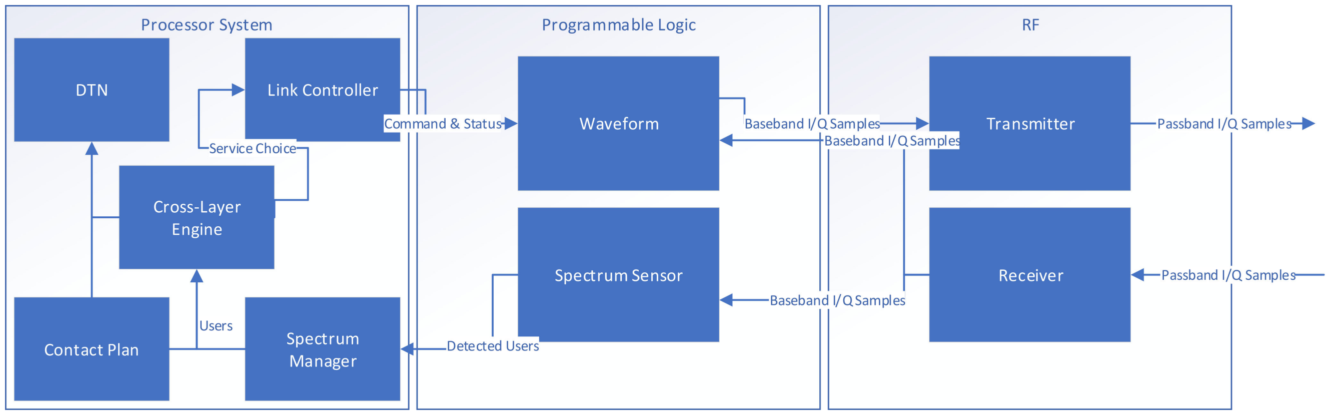

Fig. 10. SDR data flow diagram.

Event-driven processing now only contributes to sparsity and power efficiency by blocking unchanging input, but it often allows for much faster processing speeds [50].

In addition, our multi-agent approach used deep Q-learning models found in existing packages (PyTorch, RLlib). These do not exploit the advantages of SNNs, such as low power consumption and faster processing speeds. Future work may include incorporating an SNN into the reinforcement learning agent. Existing work applying SNNs to DTN routing shows an advantage over CGR methods [52], but has not yet taken a multi-agent approach.

Characteristics for an ideal lunar networking data set may be partially derived from more general data sets such as the BNNC Graph Neural Networking Challenge Data Set. A data set specific to the lunar networking environment could be developed in a similar method. The BNNC data was generated using an OMNeT++ simulation of several different network topologies. Orbital simulations such as those discussed in Section I could be used as an input to a network simulator to generate a scenario specific to the Moon, with appropriate propagation delays, data rates, and protocol stack. In addition, it is desirable for the data set to include both network layer (packet loss, packet delay) and physical layer metrics (RSSI) so that the relationship between the layers can be used to optimize the overall system. To this end we are continuing to develop higher fidelity simulation environments. Synthetic lunar data sets constructed from existing surface images have been used to train neural networks for rover localization services and surface analysis [66], [67]. The development of synthetic lunar data sets will likely be needed for many applications, including networking, due to the sparsity of available data.

B. Hardware Integration and Protocol Development

We anticipate that Internet Protocol will play a key role in developing the interface between the bundle layer and software-defined radio. Additional work is needed to reconcile traditional IP routing with the DTN bundle layer (application layer) style routing discussed in this paper. Any feasible routing method for the future lunar network should maintain compatibility with commercial providers and such networks may appear as a black box to any cognitive agent.

Additionally, to enable opportunistic routing, we envision a method of autonomously discovering when neighboring nodes are in range. This discovery must be handled first at the physical layer, and then may trigger a network layer protocol to advertise connectivity information. DTN IP Neighbor Discovery (IPND) is an Internet-Draft protocol outlining the mechanism for previously unknown DTN nodes to exchange connectivity information to allow them to begin to communicate with one another [68]. This method uses discovery beacons consisting of small UDP datagrams that allow nodes to advertise their existence to one another. Any node using an IP-based convergence layer may participate in the discovery process. This is done as part of the Bundle Protocol overlay concept, so that heterogeneous networks may still exchange data using the commonality of the bundle layer and IP underlay. In addition to the IPND concept, we may wish to investigate discovery mechanisms used in terrestrial protocols.

Perhaps the most pressing issue related to opportunistic space networks is the hardware considerations required to enable this technology. The first challenge is that the physical layer must be able to detect compatible frequencies. We feel this can be addressed with the spectrum sensing technology described throughout this paper. The next consideration is the ability to configure the software-defined radio with an appropriate waveform as needed. Finally, these dynamic algorithms must be paired with appropriate antenna technology that is capable of supporting the flexibility to autonomously track multiple nodes. We feel all of these challenges are frequently overlooked in much of the literature related to DTN routing algorithms and that this integration of the physical and network layers holds much potential for future research.

C. Simulation and Modeling

The major components described in this paper largely exist as separate technologies, with varying levels of maturity. Additional simulation and modeling are needed to integrate the DTN bundle agent, algorithmic methods, software-defined radio, and spectrum sensor into a single system. Network emulation and hardware-in-the-loop testing will be needed to fully evaluate the system. In addition, the lunar scenario should be carefully considered to develop realistic parameters for testing. Additional work is needed to develop a realistic network model with a sufficient definition of the cognitive (or lack of) capabilities of each network asset. This particular consideration has a significant impact on the paradigm of learning mechanisms that may be feasible for the network.

VII. Conclusion

In this work, we take a further step towards defining and implementing a cognitive networking system. We discussed several major fields of study related to cognitive communication for the future lunar network including: the network scenario and orbital modeling, delay tolerant networking, relevant algorithms (spiking neural networks and multi-agent reinforcement learning), spectrum sensing, and system architecture. In addition, we discuss an overview of publicly available data sets for wireless networking, several simulation tools (SNNTorch, PyTorch, and Ray), and preliminary simulation results. In future work, we hope to mature beyond basic algorithmic concepts and begin to elaborate on significant technology gaps that must be addressed to enable truly autonomous, intelligent communication in the space network environment. Many challenges exist throughout all of the layers of the system, and cognition necessitates collaboration and awareness between previously disparate elements.

Acknowledgment

The authors would like to thank Dr. Janette Briones, Dr. Peter Schemmel, Dave Chelmins, Dr. Adam Gannon, Aaron Smith, and the NASA Space Communication and Navigation program for their kind support.

References

[1] Interagency Operations Advisory Group, "The future lunar communications architecture," Rep. Interagency Oper. Advisory Group Lunar Commun. Archit. Working Group, Rep. V.1.2, Feb. 1, 2020.

[2] R. Dudukovich, K. Wagner, S. Kancharla, J. Fantl, and A. Fung, "Towards the development of a multi-agent cognitive networking system for the lunar environment," in *Proc. IEEE Int. Conf. Wireless Space Extreme Environ.*, 2021, pp. 7–13.

[3] D. J. Israel *et al.*, "LunaNet: A flexible and extensible lunar exploration communications and navigation infrastructure," in *Proc. IEEE Aerosp. Conf.*, 2020, pp. 1–14.

[4] D. J. Israel, L. V. D. Cooper, K. D. Mauldin, and K. Schauer, "LunaNet: A flexible and extensible lunar exploration communications and navigation infrastructure and the inclusion of SmallSat platforms," in *Proc. SmallSat Conf.*, 2020, pp. 1–7.

[5] S. Burleigh, "Contact graph routing," IETF, Internet-Draft draft-burleigh-dtnrg-cgr-00, 2009. [Online]. Available: https://tools.ietf.org/html/draft-burleigh-dtnrg-cgr-00

[6] D. Y. Stodden and G. D. Galasso, "Space system visualization and analysis using the satellite orbit analysis program (SOAP)," in *Proc. Aerosp. Appl. Conf.*, Mar. 1995, pp. 369–387.

[7] D. E. Lee, "Gateway destination orbit model: A continuous 15 year NRHO reference trajectory," Nat. Aeronaut. Space Admin., Washington, DC, USA, Johnson Space Center, Houston, TX, USA, Rep. JSC-E-DAA-TN72594, Aug. 2019.

[8] D. Brooks, W. Eddy, S. Johnson, and G. Clark, "In-space networking on NASA's scan testbed," in *Proc. 34th AIAA Int. Commun. Satellite Syst. Conf.*, Oct. 2016, pp. 1–9.

[9] G. Clark, W. Eddy, S. Johnson, D. Brooks, and J. Barnes, "Architecture for cognitive networking within NASA's future space communications infrastructure," in *Proc. 34th AIAA Int. Commun. Satellite Syst. Conf.*, Oct. 2016, pp. 1–10.

[10] I. O'Neill. "NASA Confirms New SIMPLEx Mission Small Satellite to Blaze Trails Studying Lunar Surface." 2020. [Online]. Available: https://www.jpl.nasa.gov/news/nasa-confirms-new-simplex-mission-small-satellite-to-blaze-trails-studying-lunar-surface

[11] M. Tsay, J. Frongillo, K. Hohman, and B. K. Malphrus, "LunarCube: A deep space 6U CubeSat with mission enabling ion propulsion technology," in *Proc. 29th Annu. AIAA/USU Conf. Small Satellites*, 2015, pp. 1–16.

[12] B. L. Ehlmann *et al.*, "Lunar trailblazer: A pioneering small satellite for lunar water and lunar geology," in *Proc. 52nd Lunar Planetary Sci. Conf.*, 2021, p. P060-02.

[13] S. Rowe *et al.*, "Lunar volatile and mineralogy mapping orbiter (VMMO): Viable science from lunar CubeSats," in *Proc. Small Satellite Conf.*, 2021, pp. 1–11.

[14] J. Zakrajsek *et al.*, "Exploration rover concepts and development challenges," in *Proc. AIAA Space Exploration Conf.*, vol. 1, Apr. 2005, pp. 1–23.

[15] R. Lent, R. Dudukovich, A. Gannon, and R. Short, "Applying the cognitive space gateway to swarm topologies," in *Proc. IEEE Cogn. Commun. Aerosp. Appl. Workshop (CCAAW)*, 2021, pp. 1–7.

[16] S. Burleigh *et al.*, "Delay-tolerant networking: An approach to interplanetary Internet," *IEEE Commun. Mag.*, vol. 41, no. 6, pp. 128–136, Jun. 2003.

[17] A. Hylton, D. Raible, G. Clark, R. Dudukovich, B. Tomko, and L. Burke, "Rising above the cloud: Toward high-rate delay-tolerant networking in low earth orbit," in *Proc. Adv. Commun. Satellite Syst. 37th Int. Commun. Satellite Syst. Conf. (ICSSC)*, 2019, pp. 1–17.

[18] R. Dudukovich, B. LaFuente, A. Hylton, B. J. Tomko, and J. C. Follo, "A distributed approach to high-rate delay tolerant networking within a virtualized environment," in *Proc. IEEE Cogn. Commun. Aerosp. Appl. Workshop*, 2021, pp. 1–5.

[19] "High-Rate Delay Tolerant Networking." NASA Glenn Research Center. [Online]. Available: https://github.com/nasa/HDTN (Accessed: Jan. 10, 2022).

[20] R. Dudukovich, A. Hylton, and C. Papachristou, "A machine learning concept for DTN routing," in *Proc. IEEE Int. Conf. Wireless Space Extreme Environ.*, 2017, pp. 110–115.

[21] R. Dudukovich, J. Briones, A. Hylton, and G. Clark, "Microservice architecture for cognitive networks," in *Proc. IEEE Int. Conf. Wireless Space Extreme Environ.*, 2020, pp. 39–44.

[22] R. Dudukovich, G. Clark, and C. Papachristou, "Evaluation of classifier complexity for delay tolerant network routing," in *Proc. IEEE Cogn. Commun. Aerosp. Appl. Workshop*, 2019, pp. 1–7.

[23] R. Dudukovich, "Application of machine learning techniques to delay tolerant network routing," Ph.D. dissertation, Dept. Electr. Eng. Comput. Sci., Case Western Reserve Univ., Cleveland, OH, USA, 2019.

[24] M. S. Net and S. Burleigh, "Evaluation of opportunistic contact graph routing in random mobility environments," in *Proc. 6th IEEE Int. Conf. Wireless Space Extreme Environ. (WiSEE)*, 2018, pp. 183–188.

[25] G. Araniti *et al.*, "Contact graph routing in DTN space networks: Overview, enhancements and performance," *IEEE Commun. Mag.*, vol. 53, no. 3, pp. 38–46, Mar. 2015.

[26] S. Burleigh, C. Caini, J. J. Messina, and M. Rodolfi, "Toward a unified routing framework for delay-tolerant networking," in *Proc. IEEE Int. Conf. Wireless Space Extreme Environ. (WiSEE)*, 2016, pp. 82–86.

[27] C. Caini and R. Firrincieli, "Application of contact graph routing to LEO satellite DTN communications," in *Proc. IEEE Int. Conf. Commun. (ICC)*, 2012, pp. 3301–3305.

[28] R. Dudukovich and C. Papachristou, "Delay tolerant network routing as a machine learning classification problem," in *Proc. NASA/ESA Conf. Adapt. Hardw. Syst. (AHS)*, 2018, pp. 96–103.

[29] M. A. L. Silva, S. R. de Souza, M. J. F. Souza, and A. L. C. Bazzan, "A reinforcement learning-based multi-agent framework applied for solving routing and scheduling problems," *Expert Syst. Appl.*, vol. 131, pp. 148–171, Oct. 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0957417419302866

[30] E. Birrane, S. Burleigh, and N. Kasch, "Analysis of the contact graph routing algorithm: Bounding interplanetary paths," *Acta Astronautica*, vol. 75, pp. 108–119, Jun./Jul. 2012. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0094576512000288

[31] J. K. Gupta, M. Egorov, and M. Kochenderfer, "Cooperative multi-agent control using deep reinforcement learning," in *Autonomous Agents and Multiagent Systems*, G. Sukthankar and J. A. Rodriguez-Aguilar, Eds. Cham, Switzerland: Springer Int., 2017, pp. 66–83.

[32] R. Lowe, Y. Wu, A. Tamar, J. Harb, P. Abbeel, and I. Mordatch, "Multi-agent actor-critic for mixed cooperative-competitive environments," in *Proc. 31st Int. Conf. Neural Inf. Process. Syst.*, 2017, pp. 6382–6393.

[33] K. Zhang, Z. Yang, and T. Başar, "Multi-agent reinforcement learning: A selective overview of theories and algorithms," 2021, *arXiv:1911.10635*.

[34] D. Mukhutdinov, A. Filchenkov, A. Shalyto, and V. Vyatkin, "Multi-agent deep learning for simultaneous optimization for time and energy in distributed routing system," *Future Gener. Comput. Syst.*, vol. 94, pp. 587–600, May 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167739X18309087

[35] V. Mnih *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, pp. 529–533, Feb. 2015.

[36] X. Li, X. Hu, W. Li, and H. Hu, "A multi-agent reinforcement learning routing protocol for underwater optical sensor networks," in *Proc. IEEE Int. Conf. Commun. (ICC)*, 2019, pp. 1–7.

[37] V. Mnih *et al.*, "Asynchronous methods for deep reinforcement learning," in *Proc. 33rd Int. Conf. Mach. Learn.*, 2016, pp. 1928–1937.

[38] Y. Zhang and M. M. Zavlanos, "Distributed off-policy actor-critic reinforcement learning with policy consensus," in *Proc. CDC*, 2019, pp. 4674–4679.

[39] D. J. Gormley, "A low-memory spectral-correlation analyzer for digital QAM-SRRC waveforms," M.S. thesis, Dept. Electr. Eng., Cleveland State Univ., Cleveland, OH, USA, May 2021.

[40] D. J. Gormley and A. A. Stock, "A spectrum sensor for CubeSat radios," in *Proc. 3rd Bi-Annu. IEEE Cogn. Commun. Aerosp. Appl. Workshop (CCAAW)*, Cleveland, OH, USA, Jun. 2021, pp. 1–5, doi: 10.1109/CCAAW50069.2021.9527303.

[41] "5 steps to an AI proof of concept," Intel Corp., Santa Clara, CA, USA, Rep. 337357-001EN, Mar. 2018. [Online]. Available: https://www.intel.com/content/dam/www/public/us/en/documents/white-papers/five-steps-ai-proof-of-concept-whitepaper.pdf

[42] "Beyond Today's AI: Neuromorphic Computing." Intel Corp. [Online]. Available: https://www.intel.com/content/www/us/en/research/neuromorphic-computing.html (Accessed: Oct. 20, 2021).

[43] C.-K. Lin *et al.*, "Programming spiking neural networks on Intel's Loihi," *Computer*, vol. 51, no. 3, pp. 52–61, Mar. 2018.

[44] T. M. Taha, C. Yakopcic, N. Rahman, T. Atahary, and S. Douglass, "Cognitive domain ontologies: HPCs to ultra low power neuromorphic platforms," in *Proc. Neuro-Inspired Comput. Elements Workshop*, 2020, pp. 1–3. [Online]. Available: https://doi.org/10.1145/3381755.3381781

[45] D. Raychaudhuri *et al.*, "Overview of the ORBIT radio grid testbed for evaluation of next-generation wireless network protocols," in *Proc. IEEE Wireless Commun. Netw. Conf.*, vol. 3, 2005, pp. 1664–1669.

[46] J. Burgess *et al.*, Sep. 2008, "The Umass/Diesel Dataset (v. 2008-09-14)," RAWDAD. [Online]. Available: https://crawdad.org/umass/diesel/20080914

[47] J. Suárez-Varela *et al.*, "The graph neural networking challenge: A worldwide competition for education in AI/ML for networks," 2021, *arXiv:2107.12433*.

[48] K. Rusek, J. Suárez-Varela, P. Almasan, P. Barlet-Ros, and A. Cabellos-Aparicio, "Routenet: Leveraging graph neural networks for network modeling and optimization in SDN," *IEEE J. Sel. Areas Commun.*, vol. 38, no. 10, pp. 2260–2270, Oct. 2020, doi: 10.1109/JSAC.2020.3000405.

[49] M. Davies *et al.*, "Loihi: A neuromorphic manycore processor with on-chip learning," *IEEE Micro*, vol. 38, no. 1, pp. 82–99, Jan./Feb. 2018.

[50] J. K. Eshraghian *et al.*, "Training spiking neural networks using lessons from deep learning," 2021, *arXiv:2109.12894*.

[51] B. Latifa, Z. Gao, and S. Liu, "Distributed multi-agent Q-learning for joint channel allocation and power control in cognitive radio networks," *J. Comput. Inf. Syst.*, vol. 8, pp. 7071–7078, Sep. 2012.

[52] R. Lent, "Adaptive DTN routing: A neuromorphic networking perspective," *IEEE Trans. Cogn. Commun. Netw.*, vol. 7, no. 3, pp. 871–880, Sep. 2021.

[53] W. Cui and W. Yu, "Scalable deep reinforcement learning for routing and spectrum access in physical layer," *IEEE Trans. Commun.*, vol. 69, no. 12, pp. 8200–8213, Sep. 2021.

[54] Y.-H. Chang, T. Ho, and L. P. Kaelbling, "Mobilized ad-hoc networks: A reinforcement learning approach," in *Proc. Int. Conf. Auton. Comput.*, 2004, pp. 240–247.

[55] M. Zhou *et al.*, "Factorized Q-learning for large-scale multi-agent systems," in *Proc. 1st Int. Conf. Distrib. Artif. Intell.*, Oct. 2019, pp. 1–7, doi: 10.1145/3356464.3357707.

[56] Z. Shou, X. Chen, Y. Fu, and X. Di, "Multi-agent reinforcement learning for Markov routing games: A new modeling paradigm for dynamic traffic assignment," *Transp. Res. C, Emerg. Technol.*, vol. 137, Apr. 2022, Art. no. 103560. [Online] Available: https://doi.org/10.1016/j.trc.2022.103560.

[57] D. Ye, M. Zhang, and Y. Yang, "A multi-agent framework for packet routing in wireless sensor networks," *Sensors*, vol. 15, pp. 10026–10047, May 2015.

[58] J. N. Foerster, G. Farquhar, T. Afouras, N. Nardelli, and S. Whiteson, "Counterfactual multi-agent policy gradients," 2017, *arXiv:1705.08926*.

[59] G. Brockman *et al.*, "OpenAI gym," 2016, *arXiv:1606.01540*.

[60] P. Moritz *et al.*, "Ray: A distributed framework for emerging AI applications," 2017, *arXiv:1712.05889*.

[61] J. Foerster *et al.*, "Stabilising experience replay for deep multi-agent reinforcement learning," in *Proc. 34th Int. Conf. Mach. Learn.*, 2018, pp. 1146–1155.

[62] M. Tan, "Multi-agent reinforcement learning: Independent vs. cooperative agents," in *Proc. 10th Int. Conf. Mach. Learn.*, 1993, pp. 330–337.

[63] J. Boyan and M. Littman, "Packet routing in dynamically changing networks: A reinforcement learning approach," in *Advances in Neural Information Processing Systems*, vol. 6, J. Cowan, G. Tesauro, and J. Alspector, Eds. San Mateo, CA, USA: Morgan-Kaufmann, 1994. [Online]. Available: https://proceedings.neurips.cc/paper/1993/file/4ea06fbc83cdd0a06020c35d50e1e89a-Paper.pdf

[64] Y. Kang, X. Wang, and Z. Lan, "Q-adaptive: A multi-agent reinforcement learning based routing on dragonfly network," in *Proc. 30th Int. Symp. High-Perform. Parallel Distrib. Comput.*, 2021, pp. 189–200. [Online]. Available: https://doi.org/10.1145/3431379.3460650

[65] M. V. Koch and J. A. Downey, "Interference mitigation using cyclic autocorrelation and multi-objective optimization," Nat. Aeronaut. Space Admin., Washington, DC, USA, Glenn Res. Center, Cleveland, OH, USA, Rep. NASA/TM–2019-220226, Jul. 2019.

[66] B. Wu, W. K. PotterRoss, P. Ludivig, A. S. Chung, and T. Seabrook, "Absolute localization through orbital maps and surface perspective imagery: A synthetic lunar dataset and neural network approach," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, 2019, pp. 3262–3267.

[67] R. Pessia, 2019, "Artificial Lunar Landscape Dataset," Kaggle. [Online]. Available: https://www.kaggle.com/romainpessia/artificial-lunar-rocky-landscape-dataset

[68] D. Ellard, R. Altmann, and A. Gladd, "DTN IP neighbor discovery (IPND)," IETF, Internet-Draft draft-irtf-dtnrg-ipnd-02, Nov. 2012. [Online]. Available: https://tools.ietf.org/html/draft-irtf-dtnrg-ipnd-02

Rachel Dudukovich received the Ph.D. degree in computer engineering from Case Western Reserve University. She is the cognitive network lead for NASA's Cognitive Communications Project. Her research and development efforts are focused on machine learning-based networking algorithms for space communications.

Dylan Gormley is currently pursuing the Ph.D. degree with Case Western Reserve University. He is the cognitive radio lead for NASA's Cognitive Communications Project. His research and development efforts are focused on hardware-based signal processing algorithms for space communications.

Shilpa Kancharla is currently pursuing the master's degree in computer science with North Carolina State University. Her interests include artificial intelligence, computer vision, robotics, embedded systems, and space technology.

Katherine Wagner is currently pursuing the Ph.D. degree in computer science with the University of Illinois–Chicago. Her interests include artificial intelligence, multi-agent decision making, and cognitive research.

Robert Short received the Ph.D. degree in mathematics from Lehigh University in 2018. He worked as a Visiting Assistant Professor of Mathematics with John Carroll University until he joined the Secure Networks, System Integration and Test Branch with NASA Glenn Research Center in 2020. He is currently focus on the foundations of networking theory and how to efficiently route data through a network using local information. His research interests lie in the intersection of abstract mathematics and real-world applications.

David Brooks is a Senior Electrical Engineer and a Software Developer with SAIC working out of NASA Glenn Research Center. His interests include software-defined radios, networking protocols, cognitive networking architectures, and delay-tolerant networking.

Jason Fantl is currently pursuing the undergraduate degree in computer science and mathematics with the University of Wyoming.

Shruti Janardhanan is currently pursuing the undergraduate degree in computer science with a focus on artificial intelligence and machine learning with the University of Massachusetts Amherst.

Alexander Fung is currently pursuing the undergraduate degree in computer science with the University of California at Berkeley, Berkeley.