Long-Range Space Data Communication Autonomous Distributed Scheduling

Gregory Howe
Stottler Henke Associates, Inc.
Colorado Springs, Colorado, USA
ghowe@stottlerhenke.com

Richard Stottler
Stottler Henke Associates, Inc.
San Mateo, California, USA
stottler@stottlerhenke.com

***Abstract*—To realize NASA's ambition of a solar system wide Internet (starting with the Earth-Moon system's LunaNet) will require optimized communications through limited data bandwidth communication. Our project, DREAMS, focuses on two aspects of optimal communications: routing and link optimization. Routing is the ability, given that links are already optimized and known (to the extent possible), to optimally schedule storage and transmission of data to maximize throughput, i.e., routing chains bundle transmissions together separated by periods of bundle storage. Link optimization is the ability to optimize a link by tuning modulation scheme, transmit power, symbol rate, etc., where optimality is determined by a weighted sum of metrics including Bit Error Rate (BER), throughput, and power consumption. The DREAMS system consists of three primary components: 1) a ML-based RF link optimizer which optimizes each RF link and apprises distributed schedulers of what data bandwidth each link can support; 2) a packet predictor which predicts and apprises distributed schedulers of the volume of future packets expected from specific applications from the daily activity schedule and the expected volume of packets not tied to the schedule; and 3) distributed schedulers, which exchange network status and schedule information with one another to continuously update and re-optimize the transmission and storage schedule.**

***Index Terms*—Scheduling, Distributed, Artificial Intelligence, Delay Tolerant Network, Routing, Optimization**

I. Introduction

NASA's Space Communication and Navigation (SCaN) program seeks to manage, maintain, and expand NASA's main networks, the Near Space Network (NSN) and Deep Space Network (DSN) with one system. Integrating and expanding these three diverse networks will require building a scalable, flexible, and efficient software and space communications infrastructure that manages heterogenous resources, e.g., the combined network will contain surface assets, Low Earth Orbit (LEO), Medium Earth Orbit (MEO), and Highly Elliptical Orbit (HEO), lunar, and Lagrange Orbit satellites, all of which have varying capabilities and may produce, store, relay, or consume network traffic. Efficiently utilizing these expensive resources will require a computationally efficient scheduling and routing algorithm capable of maximizing throughput over a network with diverse hardware capabilities.

The network managed by SCaN will include a variety of nodes ranging from CubeSats to the International Space Station (ISS). These heterogenous node types will be characterized by dramatically different computational and network capabilities. Specifically, these nodes may have differing numbers of antennas with varying Tx/Rx capabilities (e.g., transmit-only, receive-only, half-duplex, full-duplex, etc.), communicate on different frequency bands, have varying data rates, different types of antennas (e.g., omni-directional, phased-array, etc.), etc. For example, CubeSats may have one comparably low bandwidth antenna and meager computation capabilities while the ISS has multiple high-bandwidth antennas and plenty of processing power.

Additionally, space communication networks span vast distances and numerous satellites, which will necessitate multi-hop communications and routing (communications that are sent to one or more intermediary nodes). Furthermore, these networks are typically characterized by several issues including intermittent links, high latencies, low bandwidths, comparatively high bit error rates, ad hoc connections, and asymmetric data rates [1], which can vary depending on time, node, etc. These issues can be divided into two categories, predictable and unpredictable aspects, e.g., line-of-sight based link disruptions and baseline network traffic are typically predictable while sporadic space weather such as solar flares, solar wind, Coronal Mass Ejections, and unscheduled network traffic are not. An optimal scheduling/routing algorithm should be able to exploit knowledge of the predictable aspects while being able to dynamically adapt to the unpredictable aspects.

Software-Defined Radios (SDRs), which will be incorporated into upcoming missions to the Moon, Mars, and beyond provide tunable parameters that can be used to manipulate the transmit frequency, modulation scheme, etc., provide an opportunity to improve Quality of Service (QoS). Specifically, an agent can vary these parameters to maximize quantities like bandwidth while minimizing quantities such as power usage, Bit Error Rate (BER), etc. In the future, more satellites will utilize SDRs, which will further necessitate intelligent parameter tuning.

We present the Delay/disruption tolerant REinforcement learning and Aurora based coMmunication System (DREAMS), a scheduling algorithm that addresses the issues inherent to communications between satellites. The DREAMS distributed scheduling algorithm addresses these inherent issues by 1) being adjustable to differing levels of computational abilities via tunable hyperparameters; 2) making minimal assumptions about the underlying network hardware, which enables DREAMS to accommodate diverse hardware and adapt to future hardware with minimal changes to the algorithm; 3) exploiting information about network topologies and scheduled and predicted traffic known ahead of time to more optimally schedule; 4) efficiently reacting to and scheduling around unpredicted network changes, e.g., unexpected network traffic and network disruptions; 5) utilizing a novel congestion-aware multi-hop routing algorithm; 6) maximizing QoS of links.

II. Related Work

A. Scheduling and Routing

Prior work has utilized visibility aware variants of Dijkstra's algorithm [2] for routing network traffic in Delay Tolerant Networks (DTN). Prior work [3] utilized multigraphs to optimally find routes between nodes with edges of varying costs under certain cost constraints, which are untenable for congestion aware routing. This work [3] additionally proposed methods to handle scheduling in DTNs with varying levels of network knowledge but is not directly applicable to the hardware constraints managed by SCaN. DREAMS utilizes a modified version of this routing algorithm in a temporal graph to accommodate congestion costs.

B. Software Defined Radio Parameter Tuning

Prior work [4] utilized a technique for maximizing a multi-objective QoS metric for space-based communications between antennas with SDRs by varying SDR parameters over a SDR parameter space for space-based network communications. A modification of this technique [5] has been used to train an agent on real data collected from a link between Earth and a GEO satellite under clear sky conditions.

III. Methods

A. Candidate Route Finding

DREAMS uses a variant of Dijkstra's algorithm [2] applied to a temporal visibility graph to find potential low-cost routes for network traffic given visibility information, which hereon will be referred to as Viz-Dijkstra. More precisely, each node (a node typically corresponds to a satellite or ground station), v, is duplicated $\frac{T}{\Delta}$ where T is the duration of the scenario and $0 \leq \Delta T$ is a tunable time-step parameter, to create vertices $v_{t_1}, v_{t_2}, ...v_{t_{\frac{T}{\Delta}}}$

Viz-Dijkstra then uses the visibility information to add edges to our graph. Precisely, let $e\left(v^{(i)}, v^{(j)}, t_s, t_e, \delta\right)$ represent a visibility between nodes $v^{(i)}$ and $v^{(j)}$ that is available between start time t_s and end time t_e where it takes a duration of δ to send the data across the visibility. For each of these visibilities, we create the following set of edges

$$\{e(v_{t_x}^{(i)}, v_{t_y}^{(j)}) : \exists t \in [t_s, t_e - \delta) \cap [t_x, t_{x+1}), \\ (t+\delta) \in [t_y, t_{y+1})\}$$

and add them to our graph's set of edges, E. These edge's represent the ability to traverse from $v^{(i)}$ in some interval $[t_x, t_{x+1})$ to arrive at $v^{(j)}$ in some interval $[t_y, t_{y+1})$. Additionally, for each vertex v_{t_x} where $x < \frac{\Delta}{T}$ we add an edge $e\left(v_{t_x}, v_{t_{x+1}}\right)$ to represent the node's ability to store the information and transmit it using a later visibility. Note that this procedure (especially with large Δs can make multigraphs, which Dijkstra's algorithm can still be applied to with minimal changes. Lastly, Viz-Dijkstra defines a time-dependent edge cost function $C : E \times \Re^+ \to \Re^+$; the cost function defines the cost of using an edge at a specified time.

Viz-Dijkstra then applies Dijkstra's algorithm to the specified graph with the modification that Dijkstra's algorithm now stores visit times (these are computed as $t_{v_t^{(j)}} = \max\left(t_{v_t^{(j)}}, t_s^{(e)}\right) + \delta^{(e)}$) where $t_{v_t^{(j)}}$ is the visit time at $v^{(j)}$, $t_{v_t^{(i)}}$ is the visit time at $v^{(i)}$, e is the edge used, and $t_s^{(e)}$ and $\delta^{(e)}$ are the t_s and δ for edge e) which Viz-Dijkstra uses to filter the available edges in the multi-graph for the next hop (an edge should be considered if and only if $\max\left(t_{v_t^{(i)}}, t_s^{(e)}\right) < t_e^{(e)}\Big)$.

Given that the cost function has the *FIFO property* for all edges, this algorithm is guaranteed to find the minimum cost path. Additionally, when $\Delta = T$, the algorithm becomes equivalent to the shortest cost path algorithm presented by Jain, Fall, Patra [3].

When the *FIFO property* is no longer met, Viz-Dijkstra can either 1) fail to find a path when one exists or 2) return a suboptimal path. We solve this issue by shrinking Δ, which for sufficiently small Δ and regularity conditions satisfied by DREAMS's cost function, the algorithm returns the minimum cost path. DREAMS treats Δ as a hyperparameter and typically uses values such that $\frac{T}{\Delta} \leq 10$, i.e., each original node creates at most 10 nodes in the temporal graph. Additionally, DREAMS uses the cost function

$$C(e,\ t_{v_t^{(i)}}) = \alpha(t_{v_t^{(j)}} - t_{v_t^{(i)}}) + (1-\alpha)c_e$$

where $c_e \geq 0$ is a fixed cost for using the edge corresponding to expected edge congestion and $\alpha \in [0, 1]$ is a congestion/time tradeoff hyperparameter. Note that if $\alpha = 1$ the *FIFO property* is met.

DREAMS also uses the above algorithm to compute trees for its all routing setting. Specifically, the task is to route traffic originating at a vertex to some subset of vertices. This can trivially be solved inefficiently by duplicating the traffic and routing each duplicate independently. To make this routing more network efficient, one can exploit the fact that the data that needs to be delivered to each destination is the same, e.g., to deliver from A to C and D, it's sufficient to use a link from A to B (instead of twice) once then links from B to C and B to D. In a static graph (one not changing with respect to time), this is equivalent to computing the minimum Steiner Tree, which is NP-Hard. DREAMS (computationally) efficiently computes a 'good' solution to the non-static case by using Viz-Dijkstra to find low-cost paths to each node then adds all paths to a new graph and computes a low-cost rooted tree on the newly constructed subgraph.

DREAMS uses Viz-Dijkstra to find N 'fast,' M 'low congestion,' and B 'balanced' routes for a bundle using the following three procedures [1]. For the 'fast' routes, DREAMS places the entire cost weighting on the time contribution by setting $\alpha = 1$ to produce the fastest possible route. The most congested edge on the route is removed from the graph, and this process is repeated $(N-1)$ more times until N distinct routes are produced. The 'low congestion' routes are produced similarly. Specifically, DREAMS places the entire cost weighting on the congestion contribution by setting $\alpha = 0$ to find a low congestion route. The most time-consuming edge used on the route is removed, and the process is repeated $(M-1)$ more times to produce M distinct low congestion routes. The B 'balanced' routes are only produced if no 'low congestion' routes were found. DREAMS finds 'balanced' routes by binary-searching to find low values of α that return feasible routes (routes arriving before the traffic's deadline) and like the 'low congestion' procedure removes the most time-consuming edge and repeats the process $B-1$ more times to produce B 'balanced' routes.

B. Bottleneck Scheduling

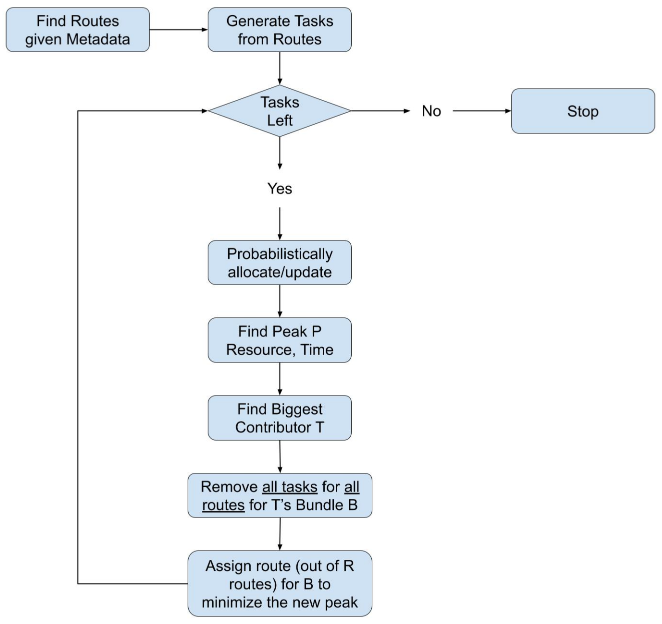
Find Routes given Metadata
Generate Tasks from Routes
Tasks Left
No
Stop
Yes
Probabilistically allocate/update
Find Peak P Resource, Time
Find Biggest Contributor T
Remove all tasks for all routes for T's Bundle B
Assign route (out of R routes) for B to minimize the new peak

Fig. 1. Bottleneck Scheduling Flowchart

Bottleneck Scheduling (BNS) [1], [6] is an algorithm that assigns tasks to resources to maximize throughput. BNS is an intuitive algorithm formulated from our conversations with human schedulers for ground-to-satellite scheduling. In the satellite-to-satellite routing domain, a resource corresponds to an antenna at a given time interval, e.g., 0 to 1 seconds, and a task corresponds to the utilization of that resource to send or receive network traffic. To this end, BNS uses the procedure detailed in Fig. 1 to assign tasks to resources to maximize the network throughput; note that this is related to but is still distinct from maximizing the total number of tasks scheduled.

In more detail, BNS first takes all routes found for the bundle and breaks them up into tasks corresponding to each hop on the route. Each task corresponds to two subtasks, a send and receive resource requirement, which correspond to the send and receive component of establishing a link. Each route is then assigned a weight p_R, which is more intuitive to think of as a probability proportional to the route's least flexible task's flexibility, i.e., tasks that are less flexible are assigned lower probabilities of being assigned. A task's flexibility is determined by the total number of available start times where the send and receive resources are available to execute the task. Each usable resource to execute the task is assigned a weighting proportional to the number of task start points that utilize the resource, which can be thought of as the probability of using the resource given that the route for this task is selected; call this weighting p_t. Note this probability can be efficiently computed in one pass by building trapezoidal like weighting structures. Lastly, we add this joint probability $p_{RT} = p_R p_T$ to the resource's congestion/allocation.

BNS then finds the peak resource (the resource with the most cumulative probability assigned to it). Next, BNS finds the task contributing the most to the peak resource, i.e., the task, T, with the greatest joint probability, p_{RT}, for the resource. BNS determines which bundle the task corresponds to and removes all congestion from all tasks corresponding to the target. If there doesn't exist a satisfiable route (a route that hasn't been disqualified due to assigning resources on previous iterations of BNS) BNS continues to the next scheduling round; otherwise, BNS considers each remaining satisfiable route and chooses a route such that the new peak plus a regularization term to discourage long routes is minimized after 'good', task assignments that locally minimize the peaks that they create, times for each of the individual tasks are selected. Note, BNS's task placement is implemented in a way that enforces that the structure of the route is maintained, e.g., if we have a two-hop route A to B, B to C, BNS ensures that the times are selected such that A to B is executed before B to C, Additionally this version of BNS supports that the sending and receiving resource utilizations may occur at different times due to high latencies in a solar system wide internet.

The new peak is calculated based on groupings on how many tasks will be disqualified by the assignment, e.g., a task A to B may disqualify a task A to C while still allowing a task B to A if both antennas A and B are full duplex. BNS considers these kinds of relationships when recomputing peaks.

The above procedure is repeated until no tasks remain. Once no tasks remain, BNS outputs a list of assigned tasks for every successfully scheduled bundle. These assigned tasks include information such as 1) when the task should be executed (and for how long), 2) which antenna pair should be used to fulfill the task/hop on the route, and 3) the meta information about the bundle. These outputs provide sufficient detail to generate an executable contact plan in High-rate Delay Tolerant Network (HDTN) [7].

BNS keeps a persistent state of the schedule including resource assignments and existing tasks, which can be augmented as new requests for scheduling come in. Furthermore, the described algorithm is very efficient, scaling near linearly (up to logarithmic multiples) with respect to additional resources and scheduling requests.

C. Distributed Scheduling

To fulfill the requirements of a solar system wide network traffic routing algorithm, DREAMS needs to be deployed and ran in a distributed manner that can efficiently respond to unexpected network traffic, outages, and conflicting schedule assignments. This is accomplished via background tasks to maintain connectivity, simple re-routing rules, and a conflict handling tool inside of BNS.

At a centralized scheduler (which can be any node in the network), DREAMS generates an initial schedule that contains periodic connectivity tasks that ensure that no node will irrecoverably be unable to contact other nodes using multiple calls to the all-routing setting scheduled via BNS. This schedule is then augmented with predicted/scheduled network activity, which is also scheduled via BNS at the centralized scheduler. This initial schedule is then propagated to the rest of the network using an initial contact plan.

Each distributed node can reschedule the remainder of the bundles route, i.e., hops in the scheduled route occurring after the data has been delivered to the node. Note, DREAMS has additional rules that limit when nodes do this rescheduling to maintain network connectivity and avoid needlessly rescheduling traffic, which is essential for minimizing miscommunications in the schedule, e.g., a receiving node not hearing about the schedule change.

Due to the distributed nature of this kind of scheduling and the need to respond to unpredicted traffic, conflicts on resources (events where a resource is over assigned in the schedule, e.g., A and B tell C that they both want to send data to it at some time) are unavoidable. DREAMS's scheduling algorithm makes efforts to reduce the frequency of these conflicts and has necessary procedures to resolve these conflicts either greedily at the node in conflict when the conflict is too close to fully resolve or via rescheduling for conflicts that are a sufficient amount of time in the future to respond to.

The described procedures under certain network conditions provides provable (in the absence of unexpected outages) guarantees for maintaining network connectivity while being able to simultaneously handle unscheduled requests. Furthermore, the procedure provides ways to reasonably respond to and schedule around unexpected outages.

D. Preliminary Results

The full DREAMS system is still in development, but we've collected some preliminary results.

For comparison of the centralized DREAMS scheduler, we implemented a semi-naïve version of the DREAMS Router. The semi-naïve version differs from DREAMS in one way—it considers only the fastest route for each bundle (i.e., the sequence of links that provide the fastest delivery time as if routing other bundles were not a concern). Note that the semi-naïve implementation still uses BNS to pick times within that route (i.e., to pick when during the visibilities on the route to actually transmit the bundle). A truly naïve algorithm would not consider congestion when picking times (instead using priority and deadline only to route bundles). We compared DREAMS's data satisfaction rate with the semi-naïve algorithm's data satisfaction rate to demonstrate and benchmark DREAMS's capabilities. When compared to a truly naïve router, DREAMS would perform even better, relatively (e.g., in a scenario where DREAMS performs 20% better than the semi-naïve scheduler, it would perform more than 20% better than a fully naïve router). Note, unlike DREAMS a truly naïve scheduler without our added procedures would also be unable to maintain network activity without a human manually generating a contact plan.

DREAMS's version of BNS has been tested on a 22 node scenario consisting of three DSN ground station; two SN ground stations; three TDRS satellites; the International Space Station; a MEO Earth satellite; the Lunar Gateway; five lunar satellites; two lunar ground stations; and four lunar rovers with a total of 360 visibilities [1]. This base scenario was scaled and modified by removing visibilites, removing nodes, and adding additional traffic. DREAMS consistently performed better than the semi-naive method achieving a 17-28% (depending on the scenario) higher data satisfaction rate than the semi-naïve baseline [1]

Since these experiments, we have incorporated a myriad of improvements to DREAMS to improve schedule quality and are in the process of collecting results of the updated DREAMS applied to distributed settings.

E. Traffic Predicition

DREAMS will augment the scheduled network traffic using two deep neural networks: one for predicting network traffic due to scheduled events and one for predicting unscheduled network traffic. By predicting this additional traffic and allocating resource utilization in the schedule, a significant portion of the scheduling can be centralized, which should result in higher quality schedules and space for future, high priority transmission can be reserved. Note that DREAMS's distributed scheduling algorithm is entirely capable of dynamically adjusting the schedule for traffic that is both unscheduled and unpredicted; the predicted traffic is solely used so DREAMS can exploit predictable information to make more holistic decisions.

The unscheduled network traffic predictor will use mined features such as the source node, target node, time of day, day of week, holiday, etc. to predict a label describing the traffic over a given time interval, i.e., the network will predict the amount of data that will be created over the time window, the expected lifetime (difference between creation time and deadline) of the data, and the priority of the data. The deep neural network will be trained to minimize the weighted sum of a joint objective function with a Mean Squared Error (MSE) loss applied to the continuous label components (amount of data and expected lifetime) and a Categorical Cross Entropy (CCE)

Loss applied to the categorical components (priority). The predicted unscheduled traffic will be scheduled by DREAMS's scheduler to allocate room for the expected traffic.

Similarly, the scheduled event network traffic predictor will be implemented as a deep neural network. The scheduled event traffic predictor will use nearly identical features to the unscheduled predictor with the addition of meta information regarding the event, e.g., event type. In addition to the information predicted by the unscheduled traffic predictor, the scheduled traffic predictor will also predict the difference between the scheduled event's scheduled start time and the event's scheduled end time (which will be incorporated into the joint objective function via MSE losses). This information will also be fed into DREAMS's scheduler.

F. Link Optimization

DREAMS will use a Reinforcement Learning (RL) based agent to maximize Quality of Service (QoS) over links in the network. DREAMS will use Distributed Distributional Deep Deterministic Policy Gradients (D4PG) [8] an improvement to Deep Deterministic Policy Gradients [9] to train an agent that uses features such as distance, relative velocity, prior SDR parameters, previous bit error rate, etc. and can fine-tune SDR parameters such as modulation scheme, bandwidth, power, etc. to minimize a customizable reward function. DREAMS's current reward function is implemented as a multi-objective, linear combination of E_b/N_0, BER multiplied by bandwidth, and Lock Losses. This RL agent is currently being trained using a high-fidelity simulator.

IV. Conclusion

Stottler Henke works on some of the world's most complex scheduling domains ranging from scheduling submarine construction to scheduling satellite-to-satellite communications with various organization. Scheduling algorithms tend to provide disproportionately large value, e.g., increasing utilization of a billion dollar's worth of managed resources by a conservative 5% would effectively deliver fifty million dollars of value. Stottler Henke has already successfully applied the proven Bottleneck Scheduling Algorithm to scheduling ground station to satellite communications and is currently applying the Bottleneck Scheduling Algorithm to satellite-to-satellite multi-hop communications and ship-to-ship multi-hop communications.

NASA's ambitions towards an Earth-Moon internet and eventually a solar system wide internet require maximizing network throughput in networks characterized by SWaP constrained satellites, intermittent links, high latencies, low bandwidths, and ad hoc connections, which traditional network procedures either struggle to or are unable to handle. To accommodate these issues, we presented DREAMS, an automated system, to greatly increase network throughput based on proven technology and Stottler Henke's prior experience with scheduling in diverse domains.

The prototype's results demonstrate the throughput improvements expected by using DREAMS and the technical feasibility of DREAMS as a centralized scheduler, and DREAMS has been expanded to handle distributed scheduling in realistic High-rate Delay Tolerant Network simulation environments and is currently being tested and evaluated. Finally, we are actively improving DREAMS and will be integrating DREAMS with existing test beds to prepare for the eventual integration with existing networks.

Acknowledgment

This material is based upon work supported by NASA under Contract No. 80NSSC22CA072. The views, opinions, and/or findings contained in this article/presentation are those of the author/presenter and should not be interpreted as representing the official views or policies, either expressed or implied, of NASA.

References

[1] R. Stottler and G. Howe, "Delay/disruption tolerant reinforcement learning aurora based communication system (dreams)," in *Proceedings of the Advanced Maui Optical and Space Surveillance Technologies Conference*, 2022.

[2] E. W. Dijkstra, "A note on two problems in connexion with graphs," in *Edsger Wybe Dijkstra: His Life, Work, and Legacy*, 2022, pp. 287–290.

[3] S. Jain, K. Fall, and R. Patra, "Routing in a delay tolerant network," in *Proceedings of the 2004 conference on Applications, technologies, architectures, and protocols for computer communications*, 2004, pp. 145–158.

[4] P. Ferreira, R. Paffenroth, A. Wyglinski, T. M. Hackett, S. Bilen, R. Reinhart, and D. Mortensen, "Multi-objective reinforcement learning for cognitive radio–based satellite communications," in *34th AIAA International Communications Satellite Systems Conference*, 2016, p. 5726.

[5] P. V. R. Ferreira, R. Paffenroth, A. M. Wyglinski, T. M. Hackett, S. G. Bilén, R. C. Reinhart, and D. J. Mortensen, "Multiobjective reinforcement learning for cognitive satellite communications using deep neural network ensembles," *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 5, pp. 1030–1041, 2018.

[6] K. Mahan, R. Stottler, and R. Jensen, "Bottleneck avoidance techniques for automated satellite communication scheduling," in *Infotech@ Aerospace 2011*, 2011, p. 1647.

[7] N. G. R. Center, "High-Rate Delay Tolerant Network." [Online]. Available: https://github.com/nasa/HDTN

[8] G. Barth-Maron, M. W. Hoffman, D. Budden, W. Dabney, D. Horgan, D. Tb, A. Muldal, N. Heess, and T. Lillicrap, "Distributed distributional deterministic policy gradients," *arXiv preprint arXiv:1804.08617*, 2018.

[9] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.