

Advancing cislunar SSA/SDA: autonomous distributed data communication scheduling and link optimization

Richard Stottler, Rakshith Mandayam, Brandon Liu, and William Lin

Stottler Henke Associates, Inc., San Mateo, CA 94402, stottler@stottlerhenke.com

ABSTRACT

Cislunar space situational awareness (SSA) and space domain awareness (SDA) will depend on the timely and reliable movement of data among heterogeneous spacecraft, sensors, relays, and ground systems operating over intermittent, bandwidth-constrained links. This paper presents an approach for autonomous data communication scheduling, routing, and link optimization intended to improve the efficiency and resilience of such networks. The scheduling approach operates in a distributed manner, allowing network nodes to coordinate schedules, propagate changes, resolve resource conflicts, maintain custody of data, and dynamically reschedule traffic in response to unexpected outages or newly arriving data. The routing and scheduling algorithms incorporate congestion awareness, support multicast communications, and use predicted future traffic to reserve network capacity and improve decisions before packets arrive. An accompanying reinforcement learning (RL)-based link optimization capability adapts communication parameters to changing link conditions. Our software, titled DREAMS (Delay/Disruption Tolerant Reinforcement Learning and Aurora Based Communication System), has been integrated with the Common Open Research Emulator (CORE) network simulator paired with the Extendable Mobile Ad-hoc Network Emulator (EMANE), a high-fidelity wireless packet transmission emulator. DREAMS has also been integrated with NASA's High-rate Delay Tolerant Network (HDTN) implementation, enabling high-fidelity evaluation using existing networking protocols and software capable of running in space. Experimental results show an increase in throughput of about 20% compared to the fastest-route baseline, as well as the ability to improve throughput when network traffic is predictable. The results demonstrate the value of combining congestion-aware routing, distributed autonomous scheduling, traffic prediction, and standards-compatible networking to support resilient data movement for future cislunar SSA/SDA architectures.

1. INTRODUCTION

Growing activity in cislunar space is increasing the need for effective space situational awareness (SSA) and space domain awareness (SDA) beyond traditional Earth-centered regimes. The National Cislunar Science & Technology Strategy identifies both extending U.S. SSA capabilities into cislunar space and establishing scalable, interoperable cislunar communications capabilities as national objectives. The U.S. Space Force similarly identifies expansion of SDA beyond geosynchronous orbit (GEO) into cislunar space as an operational requirement. As the number and diversity of spacecraft operating between the Earth and Moon increase, effective SSA/SDA will depend not only on the ability to collect observations, but also on the ability to reliably and efficiently move time-sensitive data among distributed sensors, spacecraft, relays, processing systems, and ground stations.

Cislunar communications present challenges that differ substantially from conventional terrestrial networks. Network topology changes as spacecraft move in and out of visibility; communication links may be intermittent, degraded, asymmetric, or bandwidth constrained; and nodes may differ substantially in their communication, storage, and computational capabilities. These characteristics have a predictable and an unpredictable aspect, and exploiting the predictable aspect will lead to more efficient and resilient networking. A direct end-to-end connection between a data source and its destination may not exist when the data is generated, or in fact at any point at all. Consequently, cislunar networks must make use of multi-hop communications and temporary storage at intermediate nodes via store-and-forward communication, in which a node retains data when the next communication link is unavailable and forwards it when an appropriate link becomes available. Delay/Disruption Tolerant Network (DTN) architectures and protocols address these concerns; NASA has identified DTN as a foundational capability for interoperable lunar networking, reflecting the importance of this communication model for future cislunar operations beyond SSA/SDA.

The existence of a feasible route does not imply efficient use of the network. In a network with multiple possible paths, limited visibility windows, constrained antennas, and competing data flows, routing decisions determine which network resources become congested and which remain underutilized. A strategy that independently selects the fastest available route for each transmission can repeatedly select the same desirable links and nodes, creating bottlenecks even when alternative routes have sufficient capacity. NASA's Contact Graph Routing (CGR), for example, utilizes earliest-arrival-time routing, as well as consideration of factors such as data priority and packet urgency. However, CGR does not explicitly model network-wide resource contention. In addition, such myopic schemes do not easily implement energy-aware routing (i.e., accounting for how much energy a node has and is expected to have in the future given opportunities to recharge). The routing problem is therefore a scheduling problem.

This problem becomes more consequential for SSA/SDA networks, where communication requirements may evolve rapidly as sensors and spacecraft enter and leave the network and as new information is generated. A practical cislunar SSA/SDA routing capability should accommodate frequent node entries and exits; dynamic, intermittent, degraded, asymmetric, and low-bandwidth links; heterogeneous communication technologies; and point-to-point, multicast, and broadcast traffic. It should exploit multi-hop and store-and-forward communication, adapt to temporary and permanent resource outages, and scale as the numbers of nodes, links, users, constraints, and data transmissions increase. Route diversity is also desirable so that important data can be transmitted over redundant, preferably non-overlapping paths when greater resilience is required. Finally, deterministic algorithms are valuable for mission planning, verification, and diagnosis by making identical network states and requests produce reproducible decisions.

Meeting these requirements is particularly difficult when scheduling is distributed. A centralized planner with complete network knowledge can consider competing traffic jointly, but cislunar networks cannot assume continuous connectivity to a central authority or instantaneous propagation of network state. Distributed nodes therefore need mechanisms for propagating schedule information, maintaining custody of data, detecting and resolving conflicting resource assignments, and modifying previously planned routes when unexpected traffic or outages invalidate earlier assumptions. At the same time, unnecessarily frequent replanning can itself produce inconsistencies as schedule changes propagate across long-delay or intermittently connected links. Autonomous distributed scheduling must therefore balance local responsiveness with sufficient coordination to maintain a usable network-wide plan.

Knowledge of likely future traffic can further improve these decisions. Some network traffic may be known from scheduled activities or predictable from historical behavior even before the corresponding packets are generated. Incorporating such predictions into scheduling enables anticipated traffic to influence current routing decisions and can reserve capacity that would otherwise be consumed by lower-priority or more flexible transmissions. Conversely, when traffic cannot be predicted, the distributed scheduler must accommodate newly arriving data dynamically. This paper considers both cases and evaluates how the amount of traffic known in advance affects the benefit obtained from congestion-aware scheduling.

This work presents an autonomous communication scheduling and routing approach designed to address these challenges. Candidate routes are generated using time-dependent network connectivity and include alternatives that trade off transmission time against expected congestion. A congestion-aware scheduling algorithm then jointly considers these alternatives to assign routes and communication resources while reducing network bottlenecks. The approach supports multi-hop store-and-forward routing and multicast communication and has been extended for distributed operation through schedule propagation, custody management, resource-conflict handling, and dynamic replanning following changes in network state. Predicted future traffic can be represented within the scheduler alongside traffic that has already arrived so that anticipated demand affects resource allocation. Complementing the network-level scheduling capability, a reinforcement-learning-based link optimization capability adapts communication parameters such as data rate, modulation, coding, and related parameters to changing link conditions.

The principal contributions evaluated in this paper are: (1) congestion-aware route selection and communication scheduling; (2) autonomous distributed scheduling and rescheduling in response to conflicts, newly arriving traffic, and communication-resource outages; (3) incorporation of predicted traffic into scheduling and analysis of its effect on network performance; (4) integration with the Common Open Research Emulator (CORE), the Extendable

Mobile Ad-hoc Network Emulator (EMANE), and NASA's High-rate Delay Tolerant Network (HDTN) implementation to support high-fidelity network evaluation; and (5) adaptive link optimization. The experiments use a representative cislunar network environment and demonstrate that congestion-aware routing and scheduling can substantially improve throughput relative to a baseline that selects the fastest route, while the high-fidelity integrations provide a path toward applying these capabilities to future cislunar SSA/SDA communications architectures.

2. PRIOR WORK

Our 2022 Advanced Maui Optical and Space Surveillance Technologies (AMOS) paper introduced an approach for autonomous routing and scheduling in delay- and disruption-tolerant space communication networks [4].

At the network level, the prior work treated routing as a joint route-selection and scheduling problem rather than independently selecting a shortest or fastest path for each packet. Candidate routes were generated using Viz-Dijkstra, a visibility-aware extension of Dijkstra's algorithm designed for time-dependent communication opportunities [2]. Candidate routes included alternatives emphasizing early delivery, low congestion, or a balance between the two. These alternatives were then provided to a Bottleneck Scheduling (BNS) algorithm that selected routes while considering contention for communication and storage resources across the network [3]. The intent was to distribute traffic across available network capacity rather than repeatedly select individually attractive routes that could collectively create bottlenecks. Once a route was selected, a detailed scheduler generated the timing of transmission, propagation, reception, and storage activities along that route. The scheduling representation explicitly accounted for link latency and temporary storage at intermediate nodes, allowing data to traverse multi-hop paths containing periods in which the data must be retained until a subsequent communication opportunity becomes available. This combination of route selection and detailed scheduling provided the foundation for generating executable store-and-forward communication plans subject to link, antenna, storage, and timing constraints.

The 2022 work also introduced the use of predicted future traffic during scheduling. Traffic associated with known scheduled activities could be represented as placeholder traffic before the corresponding packets were generated, allowing the scheduler to preserve capacity for anticipated transmissions rather than allocating all currently available resources to traffic already present in the network.

Finally, the prior architecture included a link optimization capability intended to exploit configurable Software-Defined Radio (SDR) parameters. Reinforcement Learning (RL) was investigated as a means of selecting communication parameters while balancing quantities such as bit error rate, achievable data rate, and power-related metrics. The paper also outlined a distributed form of the scheduling architecture in which neighboring nodes exchange schedule and network-status information and nodes may reschedule traffic passing through them as network conditions change.

The present work builds on these previously published foundations. Accordingly, the following sections do not repeat the detailed derivations of Viz-Dijkstra, Bottleneck Scheduling, or the original scheduling architecture, but instead describe the methods and capabilities needed to evaluate autonomous distributed communication scheduling, traffic prediction, multicast routing, high-fidelity network integration, and link optimization in the context considered here.

3. CONGESTION-AWARE ROUTING AND SCHEDULING

The routing and scheduling approach used in this work was introduced in our prior AMOS paper and is summarized here only to provide the context necessary for the extensions described below. Rather than selecting a route for each data packet independently, the approach treats routing and scheduling as a coupled resource allocation problem. A route that is optimal for a single packet in isolation may be undesirable when considered together with other traffic, because many independently selected routes may contend for the same communication resources. The objective is therefore to identify feasible routes that satisfy packet deadlines while distributing traffic across available network capacity and avoiding unnecessary bottlenecks. For simplicity, the centralized router/scheduler is described here first, and the distributed variant is explained in a later section.

Because routing decisions must be made very quickly and packets come in at a high rate, packets are bundled together to shorten the runtime of the scheduler/router. Scheduling packets individually provides the best results and

maximum precision at the cost of runtime complexity. Grouping packets together decreases the runtime but removes some precision. Especially when the scheduler is running in real time on compute-limited edge nodes, some grouping of packets will be necessary to provide a reasonably strong solution based on a close approximation of the traffic distribution.

Candidate routes are generated using Viz-Dijkstra, a visibility-aware modification of Dijkstra's shortest-path algorithm designed for time-dependent communication networks. The network is represented by nodes connected through visibility windows during which communication is possible. Each visibility window is characterized by properties including its availability interval, latency (based on distance), and communication bandwidth. For a particular packet, route feasibility depends not only on the sequence of nodes traversed but also on whether the packet can traverse each visibility window after reaching the corresponding node and still arrive at its destination before its deadline. This formulation naturally supports multi-hop, store-and-forward communication: a packet may arrive at an intermediate node, remain stored there while no appropriate outgoing link is available, and continue transmission during a later visibility window.

Viz-Dijkstra generates several candidate routes rather than only the single fastest route. These candidates include fast routes that emphasize earliest delivery, low-congestion routes that emphasize use of less-contended network resources, and, when needed, balanced routes that trade off these objectives. After a route is found, selected visibility opportunities on that route can be removed and the search repeated to generate distinct alternatives. The result is a set of feasible routes that provide the scheduler with alternatives having different timing and resource-utilization profiles. This distinction is important because selecting the fastest possible route for every packet can repeatedly concentrate traffic onto the same desirable links and nodes even when other feasible network capacity is available.

The candidate routes are then evaluated jointly using Bottleneck Scheduling (BNS). BNS begins by estimating the potential demand that unscheduled traffic places on the resources and time intervals available to it. Conceptually, each packet's potential utilization is distributed across its feasible alternatives rather than immediately committing the packet to one route. These allocations are accumulated across all traffic to identify resources and times that are likely to become bottlenecks. BNS then iteratively focuses on the most highly contended resource and the traffic contributing most strongly to that contention, favoring decisions for packets with relatively little scheduling flexibility. A specific route and set of resource assignments are selected to reduce the resulting resource contention peaks, after which the corresponding tentative allocations are removed and the process continues.

For routing, the consequence is that candidate routes are not judged solely according to their individual transit times. BNS considers how choosing a particular route will affect the network schedule as a whole. Packets with tight deadlines or few feasible alternatives naturally exert greater influence because their potential resource utilization is concentrated across fewer choices, whereas more flexible packets can often be shifted to less congested routes or later communication opportunities. This allows the exploitation of otherwise underutilized network capacity while preserving scarce resources for traffic that has fewer alternatives.

After BNS selects a route, the final step (titled "Aurora") constructs the detailed sequence and timing of the packet's send, transit, receive, and storage activities. Aurora explicitly represents propagation delay and storage at intermediate nodes and ensures that communication and storage resources are not overallocated. The scheduler considers antenna availability, link capacity and visibility, storage constraints, packet deadlines, and packet priorities. Priority is not used simply to schedule the highest-priority traffic first. Instead, the scheduler attempts to optimize utilization and throughput while satisfying the available traffic; when demand exceeds network capacity, priority determines which traffic is retained in the schedule. This allows the optimization process to make use of substantially more scheduling flexibility than a priority-first approach.

Together, Viz-Dijkstra, BNS, and Aurora therefore separate the scheduling problem into three related decisions: identifying feasible alternatives through a time-varying network, selecting routes while accounting for congestion and competing traffic, and producing a detailed resource-feasible transmission and storage schedule.

The implementation used in this work extends the original Viz-Dijkstra formulation by representing the network as a time-expanded graph. Rather than representing each physical network node with a single graph vertex with time-varying edges, each node is duplicated across a set of discrete time intervals. Communication edges connect time-

indexed vertices when a visibility permits data to be transmitted from one physical node to another during the corresponding interval. Additional edges connect successive temporal instances of the same physical node, explicitly representing the ability to store a packet at an intermediate node and wait for a later communication opportunity. The temporal resolution is controlled by a tunable time-step parameter, allowing a tradeoff between graph size and the fidelity with which time-dependent routing alternatives are represented.

This time-expanded representation improves path finding when routing cost depends on both transmission time and network congestion. In the earlier formulation, different paths reaching the same physical node at different times could effectively be collapsed into a single state. However, an earlier arrival at an intermediate node does not necessarily lead to the lowest-cost end-to-end route when link availability and congestion change over time. Treating different arrival-time intervals as distinct vertices preserves these alternatives during the search. More formally, the earlier Viz-Dijkstra formulation can return a suboptimal route, or fail to find an available route, when the time-dependent edge-cost function does not satisfy the First-In-First-Out (FIFO) property. The time-expanded formulation mitigates this limitation; with sufficiently fine temporal discretization, it can recover the minimum-cost path defined by the routing cost function used.

Multicast routing is supported with a straightforward extension to Viz-Dijkstra. For point-to-point traffic, the algorithm terminates once the destination node has been reached. For multicast traffic, the stopping condition is instead defined over a set of destination nodes, and the search continues until all specified destinations have been reached. This produces a routing structure that can share common portions of the path before branching toward different destinations, avoiding unnecessary duplicate transmission of identical data over the same links. Broadcast traffic is handled as a special case of multicast in which the destination set contains all nodes in the network. This approach preserves the same time-dependent visibility, deadline, and congestion considerations used for point-to-point routing while extending them to one-to-many communication.

DREAMS has also been updated with a redundant routing capability, which routes high-priority data from source S to destination D through multiple paths to increase the likelihood of timely delivery. When handling such packets, we first run the Router as described above without considering redundancy. Then, we execute a series of steps to redundantly schedule urgent high-priority packets (which we will call UHPPs for convenience). These steps are described below at a high level:

1. First, run the Route Finder and BNS without considering redundancy. Let FinalBNSState be the final state of BNS after choosing routes. Let R_i be the route chosen for UHPP i .
2. For each UHPP, create a pruned graph by removing all links in the graph from A to B if any link from A to B appears in R_i . In other words, for each link from A to B in the best route, remove links across all visibility windows from A to B in the graph. Note that there is a separate pruned graph for each UHPP.
3. For each UHPP, run Redundant Routing Dijkstra (RR-Dijkstra) (described after these high-level steps), which returns whether a route exists from S to D in the pruned graph. If a route does not exist, RR-Dijkstra also returns the minimum number m of links (individual links at a given time, not links across all time) that need to be added back to find a route as well as a set of sets of m links that will allow such a route (e.g., “ $m=2$ means two links are needed, and the adding back of the links in any of these sets would create a valid route: $\{\{\text{link2}, \text{link3}\}, \{\text{link2}, \text{link4}\}\}$ ”).
 - a. If a route exists, then find the new best routes R' on this pruned graph. **This is the set of candidate redundant routes for the UHPP.**
 - b. If a route does not exist or if too few routes were found above, we will add back in links in the following manner: Let m be the minimum number of links we must add back in to form a route and let T be the set of sets of m links which form a route (so each set in T has m links). If m is the same as the number of edges in R_i , then no redundant route exists, so stop here.
 - i. For each set S in T , we assign the set a score based on the reliability of the nodes and links in that set.
 - ii. Pick the k sets with the highest score (k is an adjustable hyperparameter). For each chosen set, add the links in this set back into the graph and run Viz-Dijkstra to find the best p routes (p is an adjustable parameter). **The collection of the $k \cdot p$ routes is the set of candidate redundant routes for each UHPP.**
4. Take FinalBNSState, which contains all 100% allocated packets. For each UHPP (which already has a chosen route in FinalBNSState from the first iteration of BNS), create a copy to be routed for redundancy. Note that while we create a copy in BNS and later in Aurora for allocating routes/resources, we only

duplicate the packet when a node sends the “original” and the “copy” to two different nodes. To this FinalBNSState, add probabilistic allocations for all of the copies of UHPPs, and run BNS as normal from here. After this step, BNS will have chosen routes for the copies as well. Call this the NewFinalBNSState.

5. Feed the NewFinalBNSState to Aurora for final scheduling.

The goal of Redundant Routing Dijkstra (RR-Dijkstra) is to return whether a route exists from S to D in a graph and, if not, return the minimum number m of links (individual links at a given time, not links across all time) that need to be added to the graph to form a route from S to D, as well as a set of sets of m links that will allow such a route, with the restriction that there is a given set of links that can be added to the graph and no others can be added. To do this, we set the weights of all links in the graph to 0, and the weights of all links that could be added from the given set to 1. Note that this only applies during the link’s periods of availability. At this point, we run Viz-Dijkstra, which finds the shortest route in the graph. If a route can be found without adding any links, it would have a cost of 0, so Viz-Dijkstra would find it (before considering any could-be-added links of cost 1).

If a route does not exist in the graph, we want to find the minimum number of links we need to add to find a route. Note that each could-be-added link has a cost of 1, so adding m links costs m . To find the minimum number of links to add, we need only find the lowest-cost route, which Viz-Dijkstra solves.

Finally, we want to return the sets (each of size m) of links that could be added to find such a route. To do this, we make one final modification to Viz-Dijkstra: Instead of stopping when finding one lowest-cost route, we run Viz-Dijkstra until it finds all lowest-cost routes. This change makes a simple alteration to the termination condition—instead of stopping when Viz-Dijkstra finds the target node, it will stop when its next route to the target node has a greater cost than any route it has already found.

4. DISTRIBUTED SCHEDULING

To fulfill the requirements of a solar system-wide network traffic routing algorithm, DREAMS needs to be deployed and run 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 with sufficient software and hardware capabilities), 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 packets’ routes, i.e., hops in the scheduled route occurring after the data has been delivered to the node. Note that DREAMS has additional rules that limit when nodes do this rescheduling, so as to maintain network connectivity and avoid needlessly rescheduling traffic, in order to minimize 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 overallocated 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’ scheduling algorithm makes efforts to reduce the frequency of these conflicts and includes 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 sufficiently far in the future.

The described procedures under certain network conditions provide 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.

Below, we describe a process for transforming a correct centralized scheduling algorithm into a distributed one and provide a proof of correctness, a proof of termination, and a proof that data packets do not cycle. The assumptions are:

1. A separate channel (or control plane) exists for transmitting schedules, traffic, and status info (this data should be less than 0.02% of data traffic).
2. All links between adjacent nodes are FIFO
3. A Correct Scheduler exists that correctly schedules tasks, routes, links, and memory storage for any portion of the scheduling problem when executed on a single node. This includes allocating (but not making any scheduling decisions regarding) resources to tasks/packets scheduled by other nodes, when it receives this information. E.g., if it is told that a link is already occupied at a specific time by a packet scheduled by another node, it will not over-allocate by scheduling one of its own packets over top of that.

Below we describe the algorithm. Note that the **S#**s are referenced later by the proofs.

S1: Each node sends two pieces of information to its immediate neighbors (immediate meaning a direct communications link currently exists):

1. Overall Schedule (route and timing of individual packets), Status (link capacity over time and Node status over time), the scheduler node's ID, the timestamp for when the schedule was created, and the timestamp for when the status was updated whenever it changes or when a new link first appears (i.e., either a planned or unplanned new line of sight has just been established).
2. Data packets at the time scheduled.

S2: Thread 1: Execute the Schedule, i.e., send packets on the appropriate immediate link according to the schedule.

S2A: To avoid a special case where a medium-priority packet can be transmitted back and forth between nodes multiple times, wasting bandwidth, and to avoid other avoidable cycles, we require node n to approve a schedule before sending node n the packets on that schedule. The schedule may be re-transmitted as part of the rescheduling process to react to changes, but that will be very low volume, and the packets will not be similarly retransmitted.

S3: Thread 2: Whenever anything changes, reschedule. Only two things can make changes:

- Schedule, congestion, and/or status changes (including new packets arriving to the network) either locally or via receiving schedule/status from adjacent nodes.
- Packet fails to show up according to the schedule (which is really a special case of #1 immediately above).

S4: Only schedule packets that originate at this node or that go through this node based on the packet's current route.

S5: Only schedule a packet if the schedule for the packet has been received from the immediately preceding node on the packet's scheduled route.

A node is free to ignore the prior routing for a packet as long as the node is on the packet's route (S4) and the schedule for the packet has been received from the immediately preceding node (S5).

The new overall schedule is sent to all immediate neighbors.

How much a node reschedules the routes/times of incoming packets is a tradeoff that involves the efficiency of the rescheduler and the CPU at the node. In a node with little CPU power, it might not reschedule ever and merely execute the routes that are given with each packet.

Next, we provide the proof of correctness and termination for the distributed scheduling algorithm.

Proof Assumptions:

- **A1:** Assume that a correct scheduler exists (i.e., a scheduler that produces correct schedules and terminates) for any portion of the scheduling problem, in isolation (i.e., that it correctly schedules a subset of transmission and storage tasks, while correctly taking into account existing allocations scheduled by other nodes).
- **A2:** Each node's scheduler terminates and produces schedules rapidly enough to have the schedule for a packet done before the packet arrives.
- **A3:** Metadata (e.g., packet size, origin, destination, priority, deadline) and schedule data (e.g., routes and times for each packet) are far smaller in volume and can be transmitted in parallel on a dedicated channel

separate from the data transmission schedule. This ensures metadata and schedule data will always arrive before the corresponding data packet.

- A4: Nodes send acknowledgements when they receive each updated schedule. Acknowledgements must be received before each corresponding data packet can be sent.
- A5: All links between adjacent nodes are FIFO
- A6: There are N nodes and L possible links.

There are two kinds of nodes: those that schedule/reschedule and those that do not. The ones that do not, merely execute the schedule and routes that they are given so they can have no effect on the correctness or the termination of the scheduling process, so these will not be further considered.

When a packet first appears at the originating node, that node correctly schedules it using its own scheduler (A1). Therefore, all packets start with a correct schedule/route.

Therefore, we only need to prove that no errors or lack of termination are introduced by the distributed coordination and rescheduling process, which includes the nodes sending updated schedules to each other asynchronously and with delays.

The proof of termination is as follows: At any moment, the number of tasks to reschedule is limited in number because the number of tasks is limited and each task is at most scheduled by the nodes on its route which must be at most N nodes. Therefore, all Schedulers will terminate.

The proof of correctness is as follows: Because of A1, there will be no mistakes in any schedule generated by any one node in isolation. I.e., all of the transmit times will be correct and there will be no resource overallocations relating to the packets scheduled by this node using the most up-to-date information that it has available. Errors could be potentially introduced by the use of stale information, however, in the form of conflicting schedules.

Separate schedules must not conflict with each other. There are only two ways schedules can conflict: 1. The same packet has different routes or times simultaneously in different schedules; 2. Two different packets are allocated to the same resource at overlapping times, causing the resource to be over-allocated. (Schedules may also conflict due to different nodes having different information about the state of the environment (e.g., a link's bandwidth, or an unexpected link downtime), but this situation always reduces to one of the above two cases, so this situation is not listed as a separate case.)

- 1) The first iteration of scheduling/routing for a packet at each node cannot produce conflicts because only nodes on the packet's route are permitted to do the schedule/routing (S4), and if none of them changes the route (i.e., reschedules), then they all accept the same route/schedule for the packet so there can be no conflicting schedule. Therefore, the only source of this type of conflict would be when a packet's route is rescheduled. Assume, for example, a route is initially created for packet P through nodes N1, N2, N3, and N4. Then assume that N2 reschedules the route for Packet P to be N1, N2, N5, and N6. However, this does not create a conflict because of S5; N3 is not allowed to schedule packet P until it gets the schedule from N2. But that schedule does not include N3 on P's route, so N3 cannot reschedule P and there is no conflict. Finally, consider the case where N2 first generates the same route for P (N1, N2, N3, N4) and sends the schedule to N3, which is therefore allowed to also schedule a route for P. Then N2 later reschedules the route for P with the new route (N1, N2, N5, N6). Therefore, there is a temporary conflict for the route of P. However, when N2 sends its new schedule to N3, at N3's next rescheduling, it has to drop N3 from the list of nodes it is allowed to schedule, and when it sends schedule information on (to N4, for example), it will show the new route for P so the change will propagate through the network. But the above shows that conflicting routes can exist temporarily at different nodes; *can this cause a problem?* No, because of S5 and A2, a correct, nonconflicting schedule for P will always arrive before P does so that the temporarily different schedules for P at different nodes have no correctness effect.
- 2) Because of A1, no single node will ever over-allocate a resource based on the information that it currently possesses. This includes its own outgoing links. So, the only possible overallocations relate to different nodes allocating packets to the same link, without each node knowing about the other. Due to status and schedule propagation latency, this can temporarily exist. However, only one node transmits a packet at a time, and since the schedule for the packet arrives before the packet arrives and necessary rescheduling completes before the packet arrives (S5 and A2), the link will not be overallocated at the time the packet

needs to be transmitted. But what about the memory capacity at the receiving node? This cannot be overallocated at the time P's transmission begins, either; because of S2A, P will not be transmitted until it receives an acknowledgement from the receiving node that the schedule it sent to the receiving node is acceptable, which includes the receiving node having enough memory capacity for the new incoming packet.

In addition, we must prove that packets will not avoidably transmit in a cycle. As described above, the schedule for the packet arrives before the packet arrives and necessary rescheduling also completes before the packet arrives (S5 and A2) so that the receiving node knows how it is going to route the packet before it sends an acknowledgment (A4). The receiving node will not send an acknowledgment if the receiving node's intended route for the packet goes through the sending node. Since the sending node will not send the packet in the first place without the acknowledgment, the potential cycle will never start.

Note that it is possible for data packets to unavoidably cycle due to unanticipated changes to the network. The most obvious case is if a link on the intended route unexpectedly goes down. This could force the node that just received the packet to transmit it back to the sender if that were the next link on the new best route. But due to speed-of-light issues, as long as status and schedule data is being transmitted as rapidly as possible (i.e., in its own channel (A3) and not waiting behind any large data packets), this is unavoidable.

5. TRAFFIC PREDICTION AND PREDICTED-TRAFFIC SCHEDULING

There are two elements to the packet predictor. The first is predicting packets associated with scheduled events. The second is predicting packets (or at least rough estimates of traffic) that are unscheduled. We implemented the predictor for scheduled and unscheduled traffic and trained it on synthetic and real data.

DREAMS predicts network traffic between nodes given scheduled events. This predicted network traffic is used to create placeholder network traffic that is passed to the DREAMS scheduling and routing algorithm. In essence, by adding placeholder traffic, DREAMS avoids using visibility windows that are needed for high-priority scheduled transmissions.

We trained a neural network to predict meta information about the event from the scheduled event with our existing pipeline. Specifically, the training pipeline works in three stages: First, it takes in a scheduled event, $e^{(i)}$. $e^{(i)}$ is mined for features, among which are the scheduled start time, scheduled end time, event type (e.g., image transmission), source station, and target station. Call the feature vector $x^{(i)}$. Second, the pipeline extracts a label from all the groups of packets that were associated with the scheduled event. The label for an event is $y^{(i)} = (y_s^{(i)}, y_d^{(i)}, y_o^{(i)}, y_t^{(i)}, y_p^{(i)})$ where $y_s^{(i)}$ is the sum of the sizes of the packets, $y_d^{(i)}$ is the delay or difference between when the event is scheduled to start and when the first packet of the event is created, $y_o^{(i)}$ is the "over" time or difference between when the event is scheduled to end and when the last packet of the event is created, $y_t^{(i)}$ is the "time to live" or average difference between when a packet is created and its deadline, and $y_p^{(i)}$ is the priority of the event. Last, we train a neural network f_θ to minimize a weighted sum of squared errors. Specifically, we train f_θ to minimize

$$\sum_{i=1}^n (y^{(i)} - f_\theta(x^{(i)}))^T \Lambda (y^{(i)} - f_\theta(x^{(i)}))$$

where Λ is a positive diagonal matrix with entries corresponding to relative importance weightings between the predictions, e.g., if $\Lambda_{11} = 1$ and $\Lambda_{22} = 2$, this would penalize the network more for mis-predicting the start time of the event than for mis-predicting the size of the event. More traditional statistical methods are possible here, but we implemented a neural network due to its versatility and ability to adapt to more complex losses in the future, i.e., predicting categorical information such as priority.

When deployed, the packet prediction module uses a separate three-stage pipeline. First, it takes in a scheduled event, $e^{(i)}$. $e^{(i)}$ is mined for a feature vector $x^{(i)}$. Second, we use our trained neural network to make a prediction $\hat{y}^{(i)} = (\hat{y}_s^{(i)}, \hat{y}_d^{(i)}, \hat{y}_o^{(i)}, \hat{y}_t^{(i)}, \hat{y}_p^{(i)}) = f_\theta(x^{(i)})$. Last, we process this prediction to create filler-packet groups for the network. Specifically, we find the predicted start $t_s^{(i)}$ and end times $t_e^{(i)}$ by adding the delay time, $\hat{y}_d^{(i)}$, to the

scheduled start time and adding the over time, $\hat{y}_o^{(i)}$, to the scheduled end time. We set the predicted number of packets, $n^{(i)}$, to be the size of the event $\hat{y}_s^{(i)}$ divided by a tunable parameter for the average size of super packets for DREAMS. Finally, we create $n^{(i)}$ super packets in equal time intervals between $t_s^{(i)}$ and $t_e^{(i)}$ with a deadline $\hat{y}_t^{(i)}$ after they are created. These super packets are all assigned the same priority $\hat{y}_p^{(i)}$ and are written to a JSON file, which is used by DREAMS to allocate placeholders in the scheduler.

After the 2022 AMOS publication, we gained access to real data and were able to train our network to predict packets more accurately for real, scheduled events. In addition, we were able to add (and refine existing) features for use in our predictor, for example, lunar position. We experimented with predicting more detailed information about each event.

We also added the ability to predict packets that do not correspond to any scheduled events. This can be viewed as predicting the background network traffic between stations after removing all the packets belonging to scheduled events. We tried and compared a handful of methods of doing this.

For unscheduled predictions, we implemented a neural network that uses the source location, the target location, and other features that we have had success with such as time of day, time of year, proximity to a holiday, space weather, etc. to predict the aggregated (over time) background network traffic generated at the source location that is transmitted to the target location. The details for training and prediction when deployed were identical to those of scheduled packet prediction with one minor difference: the aggregation was over time instead of events, e.g., the network predicted the total traffic (and metadata) for packets generated at a source that would be transmitted to a target during a specific time interval. This method has a few major benefits. First, its input size scales linearly as we add more nodes to the network. Second, it is data efficient. If we trained a network for each individual source-target pair, for most pairs (except between hubs), the data would be very limited, which would result in overfitting or needing to use smaller models. The architecture should be able to leverage relationships learned from data from different source-target pairs. Specifically, given two source-target node pairs (s, t) , (u, v) , the neural network approach can leverage relationships learned from traffic between s and t to predict better traffic between u and v . Third, the model can be trained/evaluated on select pairs of inputs, e.g., it does not need to compute traffic for all the $O(|V|^2)$ node pairs. Fourth, the model can be trained on any differentiable user-specified loss, which is often not the case for traditional statistical time series models. Finally, the model can learn complicated non-linear relationships between our features.

6. LINK OPTIMIZATION

Deep Q-Networks (DQNs) have been successfully applied to optimize communication links by varying parameters such as modulation scheme, encoding rate, symbol rate, and roll-off factor. Although this technique has shown promising results and can handle continuous state spaces, it assumes a discretized action space, which is typically not the case in link optimization—encoding rate, symbol energy, and symbol rate are not necessarily discrete. This is usually addressed by discretizing each continuous sub-action, e.g., discretizing the symbol rate. For example, consider an action a , which takes values in an interval $[0, 1]$, e.g., the percentage of our maximum transmit power that we want to use. This action is necessarily continuous but can be approximated with discrete actions, e.g., set power to 0.25 or 0.75. DQNs discretize in this fashion, which often loses finer-grained control over continuous actions. Additionally, DQNs require an output node for every possible combination of actions—more precisely, every possible combination of actions that one would want to consider. This results in an exponentially increasing action space as we add additional parameters; for example, if we later wanted to add the ability to specify an encoding scheme where we have five possible values, this addition would result in needing five times more output nodes. These issues are either solved or reduced by a newer method called Deep Deterministic Policy Gradient (DDPG) [1]. Specifically, DDPG can propose continuous actions, which results in finer-grained action control and removes the exponential increase in output nodes.

At a high level, DDPG trains an actor neural network to propose continuous actions or parameters that maximize expected cumulative reward over a series of interactions with the environment. A critic neural network approximates the action-value function $Q(s, a)$, estimating the expected cumulative reward from taking action a in state s and then following the actor’s policy. For link optimization, the reward is typically some combination of low transmit power, low bit error rate, and high bit rate. Note that DDPG allows for this reward to be fully customizable and does not

require explicit knowledge of how to calculate the reward from a state and action pair; instead, it can use external factors related to the environment.

More formally, DDPG uses the deterministic Bellman Equation

$$Q^\mu(s_t, a_t) = E_{r_t, s_{t+1}}[r(s_t, a_t) + \gamma Q^\mu(s_{t+1}, \mu(s_{t+1}; \theta^\mu))]$$

where μ maps a state (this could involve space weather, previous parameter configurations, distance between the satellites, etc.) to an action; this is typically implemented as a neural network with learned parameters θ^μ . r is a reward function that takes in a state and action and outputs a reward value. Note that r is typically not a function that we know explicitly and usually involves interacting with the environment, e.g., broadcasting over a link. $\gamma \in [0, 1]$ is the future-reward discounting factor. Lastly, Q^μ is the Q function, which evaluates the expected discounted sum of rewards of performing action a_t in state s_t and using the actor μ to select actions for the remainder of the trajectory (in our scenario, a trajectory is the sequence of state-action pairs from the start of a visibility window to the end of a visibility window where states and actions may both be dependent on the actor μ). Q^μ is typically not known ahead of time, so it is approximated by another neural network specified by parameters θ^Q , which are optimized by minimizing the temporal difference error:

$$E_{s_t, a_t, r_t} [(Q(s_t, a_t; \theta^Q) - (r(s_t, a_t) + \gamma Q(s_t, a_t; \theta^Q)))^2]$$

Simultaneously, the actor network $\mu(\cdot; \theta^\mu)$ is updated by taking steps according to the policy gradient

$$E_{s_t} [\nabla_a Q(s_t, a_t; \theta^Q)|_{s=s_t, a=\mu(s_t)} \nabla_{\theta^\mu} \mu(s_t; \theta^\mu)|_{s=s_t}]$$

These updates are applied on every iteration of the DDPG algorithm. At a high level, the first step makes the critic better at guessing how well an action performed given a state is while the second step makes the actor better at producing actions that the critic thinks are good. If the critic is accurate at gauging the quality of actions and the actor is good at producing actions that the critic thinks are good, then the actor is outputting actions that result in high rewards in the environment.

We implemented and trained Distributed Distributional DDPG (D4PG), an improved variant of DDPG, which improves on DDPG to improve training stability and the quality of the actor's learned policy [6]. Additionally, we added a policy gradient-based loss to D4PG to better handle the discrete action space.

7. EXPERIMENTAL SETUP

To evaluate the communication scheduling algorithms in a higher-fidelity networking environment, the experimental testbed integrates the Common Open Research Emulator (CORE) with the Extendable Mobile Ad-hoc Network Emulator (EMANE). CORE provides the framework for constructing and executing networks composed of multiple emulated nodes, while EMANE integrates with CORE to provide wireless packet transmission emulation. Together, CORE and EMANE allow the routing and scheduling algorithms to be exercised using packet-level network behavior rather than solely through the abstract communication and resource models used internally by the scheduler. This environment supports evaluation of dynamic network connectivity, intermittent communication opportunities, and heterogeneous links representative of cislunar communication networks.

The test environment additionally incorporates NASA's High-rate Delay Tolerant Network (HDTN) implementation [5]. HDTN provides a Delay Tolerant Network software stack capable of executing store-and-forward communication across disrupted or intermittently connected networks. The scheduling system interfaces with HDTN so that routes and communication opportunities produced by the scheduling algorithms can be exercised through an operational DTN implementation. Earlier work established that the detailed scheduling output contains the information necessary to generate executable HDTN contact plans. The integration used in the present experiments extends this capability into the CORE/EMANE environment, allowing scheduled traffic to be transmitted through emulated network nodes using HDTN rather than evaluating scheduling decisions only analytically.

This combined environment provides a bridge between algorithm-level simulation and deployment in an operational space communication network. The scheduler determines how data should traverse the network and when communication resources should be used, HDTN provides the mechanisms for storing and forwarding the corresponding data, and CORE/EMANE provides the emulated network over which those communications occur. Consequently, the testbed permits end-to-end evaluation of scheduling and routing behavior under network conditions that include changing connectivity and disrupted links while using networking software intended for space-based DTN applications.

This integration represents a significant milestone because it moves the scheduling approach beyond standalone simulation and into an operational DTN software environment. In particular, integration with HDTN provides a path toward deployment on spaceflight platforms, allowing the same scheduling and routing concepts evaluated in simulation to ultimately operate within space-based communication networks.

8. RESULTS

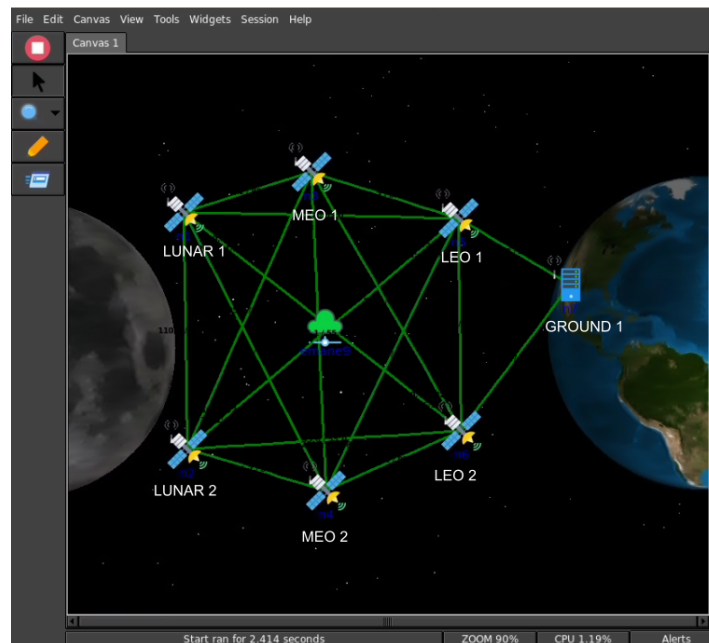


Fig. 1. CORE Network Emulator Running a Lunar Scenario with Seven Nodes, with EMANE Links.

The scenario we used for testing consists of seven nodes: one ground station node (GROUND 1), two LEO satellites (LEO 1, LEO 2), two MEO satellites (MEO 1, MEO 2), and two satellites at lunar distance (LUNAR 1, LUNAR 2). In fully distributed mode, all seven of these nodes can schedule packets. In fully centralized mode, a separate node called Scheduler is used that handles all route scheduling, which assumes perfect access to all other nodes to distribute the schedule. While real-world visibilities (defined by clear LOS between satellites) can be derived for the scenario by propagating real/contrived satellite orbits forward in time, we manually defined visibility windows that test the limits of the DREAMS scheduler. **Fig. 1** shows a screenshot of the scenario in CORE, where green lines represent EMANE wireless links. All nodes run DREAMS for scheduling and routing as well as HDTN, which we modified to interface with DREAMS. Note that the ground station cannot communicate directly with lunar or MEO satellites, thus any communication through GROUND 1 must pass through LEO satellites. All satellites are otherwise fully connected to each other (subject to visibility windows). To increase the complexity of the scenario, we periodically toggled on and off the visibilities between satellites (toggling every 10 – 20 seconds or so), preventing continuous communication between the satellites.

Although this scenario was developed in the context of NASA cislunar communications rather than a mission-specific SSA/SDA application, the underlying networking challenges are directly applicable to cislunar SSA/SDA.

In both cases, data must be moved among heterogeneous and geographically dispersed assets over intermittent, bandwidth-constrained links, often requiring multi-hop and store-and-forward communication. Nodes may enter or leave communication range, links may become temporarily unavailable, and time-sensitive data may compete for limited network resources. The specific data products and mission objectives may differ, but the underlying routing, scheduling, congestion-management, and resilience requirements are substantially the same. Thus, the scenario provides a representative environment for evaluating communication capabilities that could support future cislunar SSA/SDA architectures.

To create data to traverse through the network, we made super packets for each pair of nodes (source and destination) in the network according to a Poisson process. Thus, super packet inter-arrival times for each pair are exponentially distributed with a tunable hyperparameter λ . We generated a scenario dense enough that the fully distributed DREAMS mode is capable of scheduling $\sim 80\%$ of the data when all data is known ahead of time (fully predicted). This was the baseline scenario that we used to compare different variants of the scenario—for instance, using this scenario, we could vary the number of routes that DREAMS generated for each super packet and the number of super packets that were successfully predicted by the packet predictor to analyze scheduler performance.

For our test scenario (from which the graphics below were produced), we generated 2000 – 3000 packets, divided across 100 – 150 super packets of around 20 – 30 packets each. The packets generally had a deadline of between 70 – 220 seconds and were randomly spawned at nodes every few seconds. Each super packet was on average 300 kb, and each packet, approximately 10 – 15 kb. Note that the size of a super packet can be as large as needed; we selected values that ensured all HDTN instances and a high-fidelity radio frequency (RF) emulation ran. Each link was set at 105 kb/s; the entire scenario lasts approximately 8 – 9 minutes.

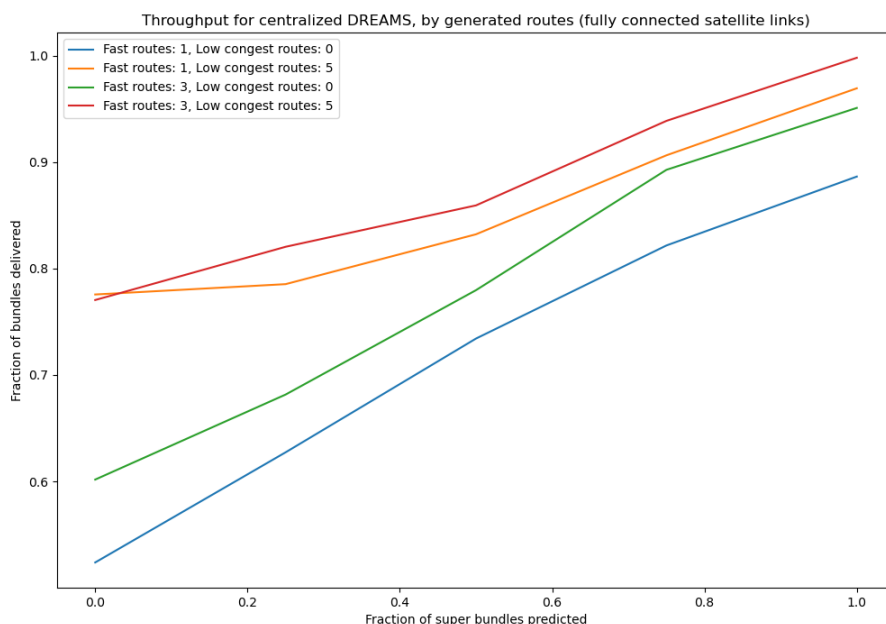


Fig. 2. Packet Delivery Performance, Centralized, Fully Connected Between Satellites.

Fig. 2 shows the performance of DREAMS running fully centralized on a fully connected visibility graph between satellites. Each line represents a different configuration of DREAMS. The X-axis represents the fraction of packets that are correctly predicted by the packet predictor. I.e., 0.5 indicates that half the packets were predicted and made known to the scheduler ahead of time, and the other half of the packets appeared randomly. The Y-axis represents the fraction of packets successfully delivered to their final destinations. There is a clear positive correlation between the ability to correctly predict packets and the successful delivery of said packets. Each line in the figure represents a different configuration of DREAMS in terms of how many routes are considered for each super packet, divided into fast and low-congestion routes. Observe that generally, the more routes DREAMS considers for each super packet, the better the schedule and packet delivery. Compared to a baseline performance of 1 route per super packet (blue

line), considering up to 8 routes (3 fast, and 5 low congestion, red line) per super packet results in a 15 – 20% increase in the performance of the schedule in the worst case (all predicted) and up to 40% increase in performance in the best case (none predicted).

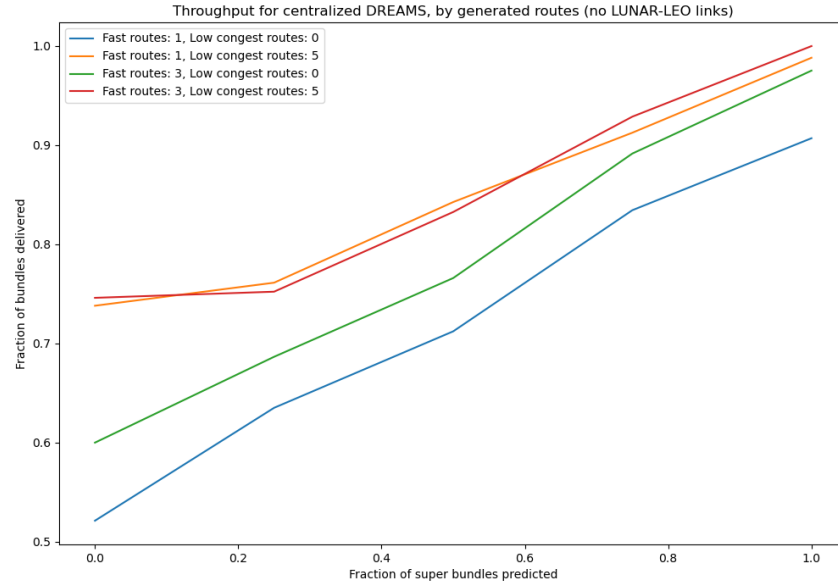


Fig. 3. Packet Delivery Performance, Centralized, No LUNAR to LEO.

Fig. 3 is very similar to Fig. 2, except the visibilities between LUNAR and LEO nodes are removed. I.e., communicating between LUNAR and LEO nodes must traverse through a MEO node. Here, we see a similar trend as in the fully connected case. A few notable differences include the following: Performance overall is very slightly worse, due to there being, in general, fewer routes to choose from. Fewer routes also result in diminishing returns as we increase the number of routes generated per super packet. There is little performance difference between generating up to six different routes versus eight—the orange line and red line are largely the same.

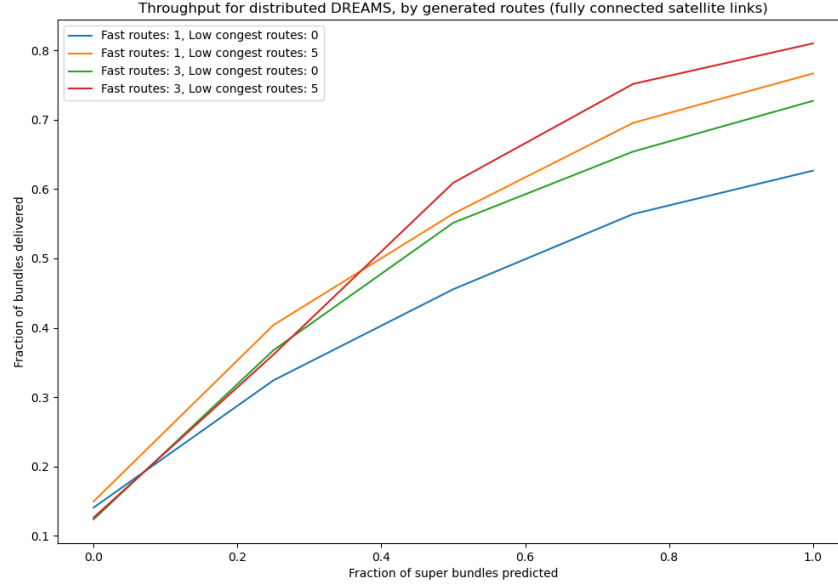


Fig. 4. Packet Delivery Performance, Fully Distributed, Fully Connected.

Fig. 4 is similar to Fig. 2, but now DREAMS is running fully distributed. This figure primarily illustrates the significance of the unmodifiable horizon on the schedule, set to a long 120 seconds in this test, particularly when fewer packets are predicted. To run DREAMS fully distributed, the unmodifiable horizon—a rolling window starting from the current time where the schedule cannot be modified—is necessary to ensure that all nodes have the latest schedule, making last-minute modifications that other nodes would not know about impossible. When packets are correctly predicted, they can be scheduled well in advance of the actual deadline, at a time when the unmodifiable horizon is also well in advance of that deadline. When packets are not predicted, these packets need to contend with the unmodifiable horizon. In the case of a particularly urgent and unpredicted packet, where the deadline is within the unmodifiable horizon, the scheduler cannot modify the schedule to include this packet, so this packet will not be delivered. Note that with an unmodifiable horizon of 120 seconds, and a packet deadline of 70 to 220 seconds, there are some super packets that are simply not schedulable. And in the best case, only ~100 seconds of the packet's lifetime can be used to deliver the packet. In Fig. 4 we see this phenomenon in action, i.e., that the performance of the scheduler's different configurations of selectable routes has little effect when most of the packets are not predicted. Any differences are largely overshadowed by the depressing effect of the unmodifiable horizon. As we increase the number of successfully predicted packets, we observe a similar pattern of performance as in the centralized case, where a larger diversity of routes allows for a better schedule. Note that this figure is just an illustrative case—with longer packet deadlines, or a shorter unmodifiable horizon, we expect the performance of the fully distributed scheduler to approach that of the centralized case. Additionally, note that in the current implementation, special tasks are hardcoded into the schedule to ensure full distribution of the schedule. This takes up space in the schedule but only does so with a constant factor.

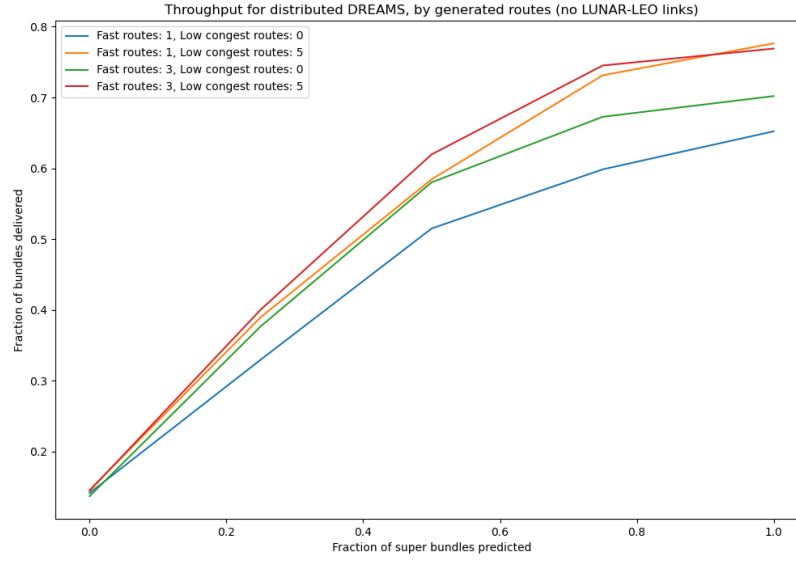


Fig. 5. Packet Delivery Performance, Fully Distributed, No LUNAR to LEO.

Fig. 5 shows a similar trend caused by the same unmodifiable horizon as Fig. 4, except without the LUNAR to LEO links. Note that removing those links causes a similar effect to that in Fig. 3, where the difference between selecting up to six routes and up to eight routes per super packet is minimized. We observe this by virtue of the red and orange lines being very similar.

Next, we discuss the link optimizer.

The link optimization agent was trained on a series of scenarios in KRATOS' Qradio environment. Specifically, we chose two random nodes from a list of satellites selected to have realistic orbits. Eleven of these orbits correspond to lunar orbits (including an orbit selected to emulate the Lunar Gateway) and six correspond to terrestrial orbits (including an orbit selected to emulate the International Space Station). Each orbit was defined to have a unique set of parameters, e.g., eccentricity, axis, inclination, perigee, etc.

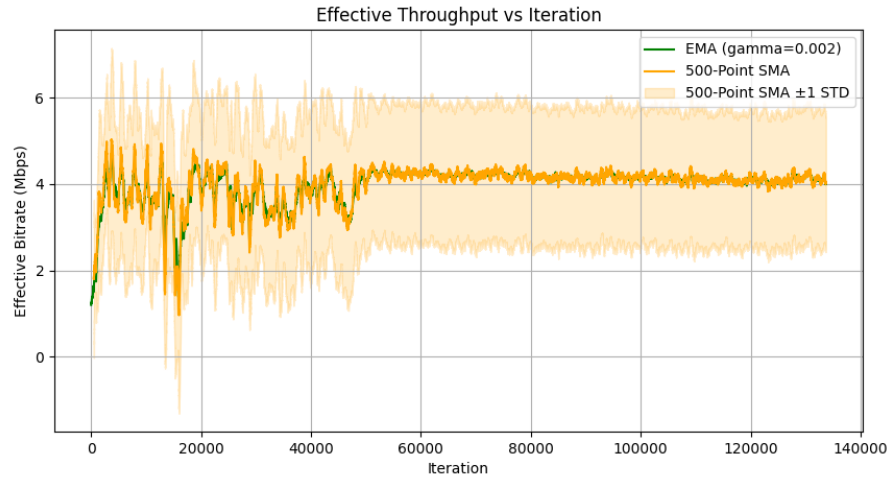


Fig. 6. Effective Throughput vs Iteration.

Above, we have the effective throughput of the agent, which is defined by the transmission bitrate multiplied by one minus the bit error rate, multiplied by one minus the percentage of time slices where data was not successfully

received. Here we see that the agent gradually improved throughout the training process for the first fifty thousand iterations, but the actual effective throughput remained noisy even after the mean throughput converged.

Similarly, we observed that as the agent converged, the bit error rate, the percentage of time slices where data was not successfully delivered, the percentage of time slices where signal lock is lost, and the successful delivery percentage, as well as the variance in all these measurements, converged significantly after around fifty thousand iterations.

We suspect that at this point, the agent learned a good combination of modulation scheme, rolloff, and frequency that contributed to a high successful delivery percentage. We also speculate that the agent learned to control its bitrate for extreme values of doppler shift, distance, angular velocity, etc. to avoid losing data in transmission.

We observed that the bitrate remained at a high variance even after all of the other fields had converged (and that the variance in bitrate explained nearly all of the variance in effective throughput after it converged). We hypothesize that this is for two main reasons. First, the optimal bitrate is likely to change as the scenario changes (this usually happens multiple times per 500-point sliding window) due to the orbits and movement changing. Because multiple scenarios go into the same window, it makes sense that the bitrate's standard deviation is high for each interval.

The link optimization agent was compared to a couple of baselines. The agent converged to achieve an effective bandwidth of approximately 4.12Mbps. We additionally set up a baseline agent by setting the modulation type to FSK, the matched filter to RRC with a rolloff factor of 0.25, the log carrier frequency to 9 GHZ (the agent learned that this was a good frequency, so we reused it in our fixed agent), and the bitrate to 7Mbps. With these settings, the agent achieved an average effective bandwidth of 2.65Mbps. We also tried 9Mbps, which was the upper bound of what the agent was allowed to specify, and which resulted in an effective bandwidth of approximately 0 (less than 0.01Mbps) due to exceptionally high bit error rate and intervals where data was not successfully sent. Note that the agent avoided these higher bandwidths, suggesting that the agent effectively learned that sufficiently high bandwidths caused issues. Last, we compared our agent to an agent that randomly selects actions from our agent's action space. This agent also displayed a low effective bitrate of 0.23Mbps. These results demonstrate that the agent was able to learn something, i.e., performed significantly better than noise and learned some complicated relationships which allowed it to avoid dangerous areas, i.e., setting bitrate around 9Mbps. Additional investigation is needed to establish stronger baselines and know to what extent these results are transferable to real settings, i.e., how our method would do in a more complicated environment and to what extent the hardware on which we are running the simulator affects results.

9. FUTURE WORK

Future work will continue to refine the routing and scheduling algorithms, including improvements to route-generation heuristics and other mechanisms for increasing schedule quality and computational efficiency. The link optimization capability can be further developed and evaluated using more diverse training and testing datasets. The packet prediction neural network approach should be compared to other approaches, especially against more data-efficient traditional statistical algorithms. More broadly, the algorithms will be exercised across a wider range of network topologies, traffic patterns, link conditions, and disruption scenarios to better characterize their robustness and scalability. Related capabilities are currently undergoing NASA flight testing, providing an opportunity to evaluate the underlying concepts in increasingly operational environments. In addition, the routing and scheduling algorithms were extended as part of a separate project optimizing the routing and MAC layers for a tactical network; the DREAMS-based optimization prototype has so far been shown by an independent evaluator to result in a 30% increase in throughput.

10. REFERENCES

- [1] Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., ... & Wierstra, D. (2015). Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*.
- [2] Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1), 269-271.
- [3] Mahan, K., Stottler, R., & Jensen, R. (2011). Bottleneck avoidance techniques for automated satellite communication scheduling. In *Infotech@ Aerospace 2011* (p. 1647).

- [4] 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*.
- [5] Glenn Research Center, "High-Rate Delay Tolerant Network." [Online]. Available: <https://github.com/nasa/HDTN>
- [6] 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.