Multigraph-based Routing in Delay Tolerant Networks: An Alternative to Contact Graph Routing

Alan Hylton
Near Space Network
NASA Goddard
Greenbelt, MD
alan.g.hylton@nasa.gov

Michael Moy
Department of Mathematics
Colorado State University
Fort Collins, CO
michael.moy@colostate.edu

Robert Kassouf-Short
Communications and Intelligent Systems
NASA Glenn
Cleveland, OH
robert.s.short@nasa.gov

Jacob Cleveland
Communications and Intelligent Systems
NASA Glenn
Cleveland, OH
jacob.a.cleveland@nasa.gov

***Abstract*—Satellites are leaving the realms of niche use, extending our day-to-day networked infrastructure to space – thereby forcing a generalization of network architectures. The Delay Tolerant Networking (DTN) protocol is being developed to give rise to this new Solar System Internet.**

Predominantly, DTNs in space use globally-distributed contact tables to compute routes. In this paper, we propose and analyze a novel optimized approach for route computations that improves upon traditional approaches. As the general DTN will always include some scheduled links, our new algorithm enables greater scalability and practicality of DTN routing.

These contact tables include windows when two nodes can communicate and were classically organized into a *contact graph*, where the vertices represent contact opportunities. Because the complexity of a contact graph grows with the number of contacts, pathfinding on it does not scale. A new structure using multigraphs with the same data is proposed. We show that a multigraph-based approach, which we call *contact multigraph routing*, exhibits performance superior to routing based on contact graphs, allowing greater scaling to schedule-based routing.

In this paper, the multigraph-based algorithm is detailed and a proof is included showing it outperforms the previous algorithm given the same input. Pseudocode is included, as are simulation results. We conclude with suggested future work.

I. Introduction

Classical space communications involves manually scheduled point-to-point links. After another record year of deployments in 2022 [1], an estimated 7,500 satellites are operational [2], pushing the need to graduate to space networking. The current effort towards a Solar System Internet is Delay Tolerant Networking (DTN) – a store, carry, and forward networking overlay. DTN routing remains under active research (see [3]–[5]), and in this paper we propose an optimized, drop-in replacement for the standard DTN routing algorithm to enable operational use, targeting NASA's LunaNet [6].

In the terrestrial Internet, a number of assumptions are made that simplify the scalability problem. Among these are end-to-end connectivity, low latency (round trip times less than seconds), and low mobility. These assumptions allow networks to be modeled by *graphs*. In space, however, these assumptions fail – therefore, more general structures must be developed to model space networks in order for routing to take place.

The current standard is to represent the network's connectivity using a *contact graph*, where vertices represent individual scheduled contacts instead of network nodes. As data can be stored between forwarding opportunities, a path can be computed using *contact graph routing* (CGR), which uses a modified Dijkstra's algorithm to optimize for delivery time. The driver for complexity in CGR is that each contact becomes a vertex in the contact graph. Hence, the algorithm does not scale well with either the size of the network or the number of contacts, limiting the number of contacts in a given schedule.

A. Contact Graph Routing

We give a brief overview of CGR and its pathfinding algorithm, following [7] (see also [8]). A *contact* between two DTN nodes is a scheduled window of time during which data can be transmitted. A contact is written as $C_{A,B}^{t_0,t_1}$, where A and B are the source and destination nodes, and t_0 and t_1 are the start and end times of the contact. In the algorithms, the source and destination of an arbitrary contact C are written as $C.src$ and $C.dst$, and the start and end times are written as $C.start$ and $C.end$. CGR depends on a globally distributed and consistent schedule of contacts between its network constituents, called a *contact plan* (CP), and includes data rates and one-way light times (OWLT) for each contact.

We place an edge between two contacts if data could be sent successively along them; that is, there is a (directed) edge from $C_{A,B}^{t_0,t_1}$ to $C_{D,E}^{t_2,t_3}$ if $B = D$ and $t_3 > t_0$. If a route from A to D is to be found starting at time t_0, a root contact $C_{A,A}^{t_0,\infty}$ and a terminal contact $C_{D,D}$ are added to the contact graph with appropriate edges. A version of Dijkstra's algorithm, optimizing for arrival time, is then used to find a path from $C_{A,A}^{t_0,\infty}$ to $C_{D,D}$, giving a path from node A to node

D in the network. This is called the Contact Graph Dijkstra Search [7], and is Algorithm 4 in the appendix.

CGR encompasses both this pathfinding algorithm and a larger strategy for generating lists of routes and queuing. In particular, CGR uses a version of Yen's algorithm to generate a list of routes to a given destination. A route for a message is then selected based on its size, the available volume of candidate routes, its time to live (TTL), etc. We show that by using an alternative data structure, a *multigraph*, the scalability of routing based on the same data is improved. This alternative approach, which we call *contact multigraph routing* (CMR), was first described in [3], and a more thorough overview was given in [9].

II. Contact Multigraph Routing

A. The contact multigraph – a multigraph model of a network

A *multigraph* is a graph that allows for multiple edges between a given pair of vertices, and the data of a contact plan can be used to define a multigraph model of a network. We will let the vertices be network nodes and let each contact $C_{A,B}^{t_0,t_1}$ be an edge from vertex A to vertex B. We call the resulting multigraph a *contact multigraph*. The time intervals may be thought of as labels on the edges, making this a labeled, directed multigraph. Note that vertices here represent the network nodes, making this model more similar to traditional graph-based models of networks than contact graphs. A similar approach was suggested in [10], which predates CGR, and the current approach was independently suggested in [3], [9] by the current authors. However, the proofs of improvement and analyses presented here were previously unrealized. Much more recently, an approach similar to ours has also been suggested in [11] based on the same multigraph model of the network. However, our approach differs in that we remain within the framework of CGR.

Dijkstra's algorithm can be used to find paths through a contact multigraph, again optimizing for delivery time. This algorithm follows the classic version of Dijkstra's algorithm but only explores through edges (contacts) that are available after the arrival time to the current vertex.

Prior to the formalities, we provide an example. Consider a network with three nodes, A, B, and D; the goal will be to send messages from A to D. Figure 1 shows a contact graph (top) and a multigraph (bottom), both modeling the same network and the same four contacts in the contact plan.

We outline the steps of the respective algorithms on these two graphs with starting time 0, source A, and destination D, starting with the traditional Contact Graph Dijkstra Search (see Algorithm 4 for pseudocode). We begin at $C_{A,A}^{0,\infty}$ and explore out each edge, finding arrival times of 0 to $C_{A,B}^{0,1}$ and 2 to $C_{A,B}^{2,3}$. Next, $C_{A,B}^{0,1}$ is chosen as the current contact as it has the earliest arrival time of the unvisited contacts, and we repeat, finding an arrival time of 4 to $C_{B,D}^{4,5}$ and 6 to $C_{B,D}^{6,7}$. Next, $C_{A,B}^{2,3}$ is chosen as the current contact since it has the next smallest arrival time of 2, and we explore its outgoing edges, finding that we cannot improve the arrival time to either $C_{B,D}^{4,5}$

$$\begin{tikzcd}
 & C_{A,B}^{0,1} \arrow[r] \arrow[dr] & C_{B,D}^{4,5} \arrow[dr] & \\
C_{A,A}^{0,\infty} \arrow[ur] \arrow[dr] & & & C_{D,D} \\
 & C_{A,B}^{2,3} \arrow[r] \arrow[ur] & C_{B,D}^{6,7} \arrow[ur] &
\end{tikzcd}$$

$$\begin{tikzcd}
A \arrow[r, bend left, "C_{A,B}^{0,1}"] \arrow[r, bend right, "C_{A,B}^{2,3}"'] & B \arrow[r, bend left, "C_{B,D}^{4,5}"] \arrow[r, bend right, "C_{B,D}^{6,7}"'] & D
\end{tikzcd}$$

Fig. 1. A contact graph and a multigraph modeling the same network.

or $C_{B,D}^{6,7}$. The next current contact would be $C_{B,D}^{4,5}$, and since it has a destination of D, we have found an optimal path with destination D, which has a best delivery time of 4.

Next, we consider the corresponding Dijkstra search in the multigraph (pseudocode is given in Algorithm 1). We begin at A at time 0, and explore through both outgoing contacts, finding an arrival time of 0 to B, arising from the contact $C_{A,B}^{0,1}$. Then B becomes the current vertex, and we explore out through each of its outgoing contacts, finding an arrival time of 4 to D through the contact $C_{B,D}^{4,5}$. Since D becomes the next current vertex, we have found an optimal path with a best delivery time of 4.

These algorithms both found the same path with the same best delivery time, but the search in the contact graph included a redundant step by using $C_{A,B}^{2,3}$ as the current contact. This step cannot improve the arrival time to either $C_{B,D}^{4,5}$ or $C_{B,D}^{6,7}$ because they were already reached from $C_{A,B}^{0,1}$, which had an earlier arrival time than $C_{A,B}^{2,3}$. If we remove this step from the search, the contacts explored in the two algorithms are in one-to-one correspondence. This suggests the search through the multigraph will generally be faster. In fact, Theorems 1 and 2 given in Section III will show that the two algorithms exhibit similar behavior on any given contact plan, thus showing that the number of steps required by the search in the multigraph will always be less than or equal to the number of steps required by the search in the contact graph.

B. The Contact Multigraph Dijkstra Search

The new pathfinding algorithm is given in pseudocode; it is split into three components: the main algorithm is Algorithm 1, which uses Algorithms 2 and 3. This mimics the presentation of the Contact Graph Dijkstra Search (Algorithm 4) given in [7] for easy comparison. For a given contact C, the one-way light time of C is written as $C.owlt$, and $owlt_{mgn}$ is a margin for one-way light times (see [7]). The predecessor $v.pred$ of a vertex v records an incoming contact and can be used at the end of the algorithm to reconstruct a route.

Algorithm 1 finds a path from v_r to v_d with best delivery time BDT (the earliest time to reach v_d). Here, a "path"

Algorithm 1 Contact Multigraph Dijkstra Search

```
Data: Contact plan $CP$, root vertex $v_r$, destination vertex $v_d$,
      $initial\_time$
Result: Route $R$ from $v_r$ to $v_d$ with best delivery time $BDT$
 1: construct vertex set $V$ from all srcs and dests of $CP$
 2: for all $v \in V$, set $v.arr\_time = \infty$, $v.visited = False$,
    $v.pred = \{\}$
 3: $v_r.arr\_time = initial\_time$
 4: $v_{curr} = v_r$
 5: while true do
 6:     $V = \text{MRP}(CP, V, v_{curr})$
 7:     $v_{next} = \text{VSP}(V)$
 8:     if $v_{next} \neq \{\}$ then
 9:         $v_{curr} = v_{next}$
10:     else
11:         break
12:     end if
13: end while
14: route reconstruction using predecessors to find $R$
15: $BDT = v_d.arr\_time$
```

Algorithm 2 Multigraph Review Procedure (MRP)

```
Data: $CP$, $V$, $v_{curr}$
Result: Revised $V$
 1: for contact $C \in CP$ such that $C.src = v_{curr}$ do
 2:     if $C.end \leq v_{curr}.arr\_time$ then
 3:         skip $C$
 4:     end if
 5:     if $C.dst.visited$ then
 6:         skip $C$
 7:     end if
 8:     $arr\_time = \max(C.start, v_{curr}.arr\_time) +$
        $C.owlt + owlt_{mgn}$
 9:     if $arr\_time < C.dst.arr\_time$ then
10:         $C.dst.arr\_time = arr\_time$
11:         $C.dst.pred = C$
12:     end if
13: end for
14: $v_{curr}.visited = True$
```

in the network means a sequence of contacts with matching endpoints such that each contact is available after the arrival time to its source vertex. The algorithm makes use of the fact that arrival times cannot decrease along a path, mimicking the reasoning of the usual Dijkstra's algorithm. Note one could modify the algorithm to find a shortest path tree reaching all destinations [7], [11].

III. Theoretical Comparison

A. Complexity

The complexity of Algorithms 1 and 4 depend on their implementation, and we have chosen to give similar pseudocode for these algorithms for easy understanding and for the later case-by-case comparison instead of optimizing.

Algorithm 3 Vertex Selection Procedure (VSP)

```
Data: $V$
Result: $v_{next}$
 1: $v_{next} = \{\}$
 2: $t_{earliest\_arrival} = \infty$
 3: for vertex $v \in V$ do
 4:     if $v.visited$ then
 5:         skip $v$
 6:     end if
 7:     if $v.arr\_time \geq v_d.arr\_time$ then
 8:         skip $v$
 9:     end if
10:     if $v.arr\_time < t_{earliest\_arrival}$ then
11:         $t_{earliest\_arrival} = v.arr\_time$
12:         $v_{next} = v$
13:     end if
14: end for
```

To analyze complexity, view a contact multigraph as a (directed) simple graph with each edge storing a list of contacts with source and destination equal to the source and destination of the edge. An example corresponding to Figure 1 is shown in Figure 2. Let $|V|$ be the number of vertices and let $|E|$ be the number of edges in the underlying simple graph. We will also assume there are no overlapping contacts for each edge[1]. That is, the time intervals for the contacts over a given edge are disjoint. For any ordering of the edges, let m_i be the *multiplicity* associated to the i^{th} edge, that is, the number of contacts in the contact plan with source and destination equal to the source and destination of the i^{th} edge. Let $|CP|$ be the total number of the contacts in the contact plan, so that $\sum_{i=1}^{|E|} m_i = |CP|$.

$$\begin{tikzcd} A \arrow[r, "\{C_{A,B}^{0,1}, C_{A,B}^{2,3}\}"] & B \arrow[r, "\{C_{B,D}^{4,5}, C_{B,D}^{6,7}\}"] & D \end{tikzcd}$$

Fig. 2. The multigraph model of a network can be viewed as a simple directed graph with a set of contacts associated to each edge. This example describes the same multigraph as in Figure 1.

Algorithm 1 can be implemented by following any implementation of Dijkstra's algorithm on the underlying simple graph. Here the optimal contact associated to an edge is the one producing the earliest arrival time, as in Algorithm 2, so it is the first contact available after the arrival time to the source vertex. The time complexity is thus equal to that of the implementation of Dijkstra's algorithm plus the time required to find the optimal contact associated to each edge. A linear search through the list of contacts associated to the i^{th} edge requires a time of $O(m_i)$. Since each edge is explored at most once over the course of Dijkstra's algorithm, these steps contribute a total time of at most $O(\sum_{i=1}^{|E|} m_i) = O(|CP|)$.

[1]Algorithms 1 and 4 are general enough to handle overlapping contacts, which could model multiple channels of communication between a given pair of nodes. These complexity calculations could be adjusted to handle this.

We can improve this bound if we use our assumption that contacts over a given edge do not overlap. Assuming the contacts for each edge are stored sorted by end time, a binary search takes $O(\log(m_i))$ for the i^{th} edge. Again, this is performed at most once for each edge, so the total time required by these steps is $O\left(\sum_{i=1}^{|E|}\log(m_i)\right)$. Observing that $\left(\prod_{i=1}^{|E|} m_i\right)^{\frac{1}{|E|}} \leq \frac{1}{|E|}\sum_{i=1}^{|E|} m_i$ by the arithmetic mean-geometric mean inequality, we can bound the time as follows:

$$\begin{aligned}\sum_{i=1}^{|E|}\log(m_i) &= \log\left(\prod_{i=1}^{|E|} m_i\right)\\ &\leq \log\left(\left(\frac{1}{|E|}\sum_{i=1}^{|E|} m_i\right)^{|E|}\right)\\ &= |E|\log\left(\frac{1}{|E|}\sum_{i=1}^{|E|} m_i\right)\\ &= |E|\log\left(\frac{|CP|}{|E|}\right).\end{aligned}$$

Therefore, the time required for determining the optimal contact for each edge is $O(|E|\log(|CP|/|E|))$. Note that the term $|CP|/|E|$ is the average multiplicity of an edge.

The remaining time required now depends on the implementation of Dijkstra's algorithm. The best known time complexity for Dijkstra's algorithm is $O\big(|E| + |V|\log|V|\big)$ (see [12]), giving an implementation of Algorithm 1 with a time complexity of $O\big(|E|\log(|CP|/|E|) + |V|\log|V|\big)$.

The complexity given above shows that Algorithm 1 scales better with the size of the contact plan than the classic Contact Graph Dijkstra Search. Indeed, since the contact graph has $|CP|$ vertices, the corresponding Dijkstra search has a time complexity of (at least) $O(|CP|\log|CP|)$, as noted in [7]. In the following section, we will elaborate on this comparison by considering the performance of the two algorithms.

B. A direct comparison of algorithms

The comparison of our Contact Multigraph Dijkstra Search (Algorithm 1) with the traditional Contact Graph Dijkstra Search (Algorithm 4) is eased through a simple modification to Algorithm 4, made in Algorithms 7 and 8, which results in essentially the same search as Algorithm 1. The main modification occurs in lines 18-20 of Algorithm 8, which marks additional contacts as visited. The justification is that no contact with the same destination as C_{curr} can achieve a better arrival time to subsequent contacts as C_{curr} was selected in the prior loop as the unvisited contact with the earliest arrival time. Thus, all contacts with the same destination should be marked as visited at the same time. Additionally, the lists of visited nodes for each contact are not recorded in the modified algorithm, so lines 8-10 and line 15 of Algorithm 5 have been removed to produce Algorithm 8. This is because the modification marking all contacts with the same destination as visited at the same time prevents loops, which was the purpose of recording lists of visited nodes, as described in [7]. The only modification found in Algorithm 7 is the use of Algorithm 8 in line 6. In particular, Algorithm 7 still uses Algorithm 6, like the original Contact Graph Dijkstra Search.

The modification results in an algorithm that, in effect, marks nodes as visited and associates arrival times to nodes rather than to contacts. Since this is the same idea behind Algorithm 1, it is not too surprising that these algorithms execute similar searches. This is made precise by the following theorem; the proof is a line-by-line comparison of the algorithms and is left to the appendix.

Theorem 1. *Given the same contact plan and routing problem, Algorithms 1 and 7 require the same number of iterations of their respective* **while** *loops, given the following: (1) the first* **for** *loops of Algorithms 2 and 8 search through the contacts in the same order and (2) the* **for** *loop of Algorithm 6 orders contacts based on their destination vertices and this order is consistent with the order of the* **for** *loop of Algorithm 3.*

Informally, this theorem shows that Algorithms 1 and 7 run in approximately the same amount of time. Compared to the original Algorithm 4, the modified Algorithm 7 marks additional contacts as visited when they cannot improve arrival times. That is, the original Algorithm 4 may perform some iterations of its **while** loop that cannot improve arrival times, which means that arrival times, visited markers, predecessors, visited node lists, BDT, and C_{fin} are left unchanged for these steps, and the current contact is marked as visited at the end. So given the same inputs, the number of iterations of the **while** loop in Algorithm 4 is greater than or equal to the number of iterations of the **while** loop in Algorithm 7. Therefore Theorem 1 implies the following, under analogous assumptions on **for** loops.

Theorem 2. *Given the same contact plan and routing problem, the* **while** *loop of Algorithm 1 requires fewer than or equal iterations compared to the* **while** *loop of Algorithm 4.*

The similarity of the loops between the two algorithms provides strong evidence that Algorithm 1 will run faster than Algorithm 4 on the same input.

IV. Experimental Comparison

For comparison, both algorithms were implemented in Python, closely following our pseudocode. Comparability between implementations took precedence over optimization.

	t_{CG}	t_{Mult}	t_{Mult}/t_{CG}
1 day	1.234	.273	.222
7 days	78.567	12.845	.163
14 days	294.699	41.843	.142

TABLE I
Results for Network 1. t_{CG} is the time in seconds taken by Algorithm 4, t_{Mult} is the time taken by Algorithm 1, and t_{Mult}/t_{CG} is included to compare the performance.

	t_{CG}	t_{Mult}	t_{Mult}/t_{CG}
1 day	.603	.110	.182
7 days	18.411	3.879	.211
14 days	75.730	15.826	.209

TABLE II
RESULTS FOR NETWORK 2. t_{CG} IS THE TIME IN SECONDS TAKEN BY ALGORITHM 4, t_{Mult} IS THE TIME TAKEN BY ALGORITHM 1, AND t_{Mult}/t_{CG} IS INCLUDED TO COMPARE THE PERFORMANCE.

	100 contacts	1000 contacts	5000 contacts
10 vertices	.308	.215	.220
30 vertices	.311	.109	.088
50 vertices	.361	.069	.058

TABLE III
RESULTS FOR RANDOMLY GENERATED NETWORKS OF VARIOUS SIZES. THE TABLE SHOWS t_{Mult}/t_{CG} IN EACH CASE WHERE t_{CG} IS THE TIME IN SECONDS TAKEN BY ALGORITHM 4 AND t_{Mult} IS THE TIME TAKEN BY ALGORITHM 1.

Using the Satellite Orbital Analysis Program (SOAP), two space networks were simulated, Network 1 and Network 2. Network 1 contains 13 vertices (satellites and ground stations) on Earth and the moon. Network 2 contains 11 vertices on Earth, the moon, and Mars. SOAP was used to create contact plans for these networks over periods of 1, 7, and 14 days, where a contact between two objects was created whenever there was a line of sight. For each contact plan, both algorithms were used to search for paths between each distinct pair of objects and with starting times on each hour for the duration of the contact plan. The times taken by the two algorithms are shown in Tables I and II, along with the ratios of the times.

Tests were also run on randomly generated networks, with various numbers of vertices and contacts. These networks were generated by choosing a random start time for each contact and a random length of the contact, with the maximum duration of a contact equal to one tenth of the total duration of the test (in both cases, a uniform distribution was used). For each choice of the number of vertices and contacts, ten different networks were generated and both algorithms were run for each pair of distinct vertices in each network. Table III shows the ratios of the times taken by the two algorithms.

As expected based on Theorem 2, Algorithm 1 outperforms Algorithm 4 in every case. We can also see that t_{Mult}/t_{CG} tends to be smaller for larger networks and contact plans, which is not surprising given the difference in the time complexities of the algorithms. These tests provide further evidence that Algorithm 1 scales better with the size of the network and contact plan.

V. CONCLUSION

As some links are very predictable and some OWLTs preclude real-time discovery, space networks will always feature scheduled components. Thus, it is necessary to improve the algorithms that compute schedule-based routes.

Classical DTNs have used CGR, and without adding any requirements a more economical data structure and its corresponding updated algorithm were proposed here. The algorithm was shown to be superior to CGR and is readily implementable from the pseudocode; see the DTN implementation HDTN [9], [13] to enhance scalability. Therefore, this product has matured DTN routing in the name of practicality.

A. Future work

1) Rewrite Yen's algorithm as used in CGR (see [7], [9]).
2) Research of the interplay of various routing "domains."
    - Solar-system scale DTNs may have several schedule-based segments which are not globally distributed. These pieces will need to be joined by a routing protocol, which could be scheduled.
    - Dynamic routing in DTNs will be limited by the practicality of feedback. How do we route between these dynamic and scheduled routing regions?
3) Data flows and priorities can now be optimized by inspecting the nature of queuing in this framework.

VI. ACKNOWLEDGEMENTS

The authors acknowledge NASA interns Olivia Freides and Brittany Story, who developed the SOAP simulations.

REFERENCES

[1] G. World, "How many satellites are orbiting around earth in 2022?" 11 2022. [Online]. Available: https://www.geospatialworld.net/prime/business-and-industry-trends/how-many-satellites-orbiting-earth/
[2] E. Space Debris Office, "Space environment statistics," 12 2022. [Online]. Available: https://sdup.esoc.esa.int/discosweb/statistics/
[3] A. Hylton, R. Short, J. Cleveland, O. Freides, Z. Memon, R. Cardona, R. Green, J. Curry, S. Gopalakrishnan, D. V. Dabke, B. Story, M. Moy, and B. Mallery, "A survey of mathematical structures for lunar networks," in *2022 IEEE Aerospace Conference*, 2022.
[4] J. Cleveland, A. Hylton, R. Short, B. Mallery, R. Green, J. Curry, D. V. Dabke, and O. Freides, "Introducing tropical geometric approaches to delay tolerant networking optimization," in *2022 IEEE Aerospace Conference*, 2022.
[5] R. Short, A. Hylton, J. Cleveland, M. Moy, R. Cardona, R. Green, J. Curry, B. Mallery, G. Bainbridge, and Z. Memon, "Sheaf theoretic models for routing in delay tolerant networks," in *2022 IEEE Aerospace Conference*, 2022.
[6] D. Baird, "Lunanet: Empowering artemis with communications and navigation interoperability," 10 2021. [Online]. Available: https://www.nasa.gov/feature/goddard/2021/lunanet-empowering-artemis-with-communications-and-navigation-interoperability
[7] J. A. Fraire, O. De Jonckère, and S. C. Burleigh, "Routing in the space internet: A contact graph routing tutorial," *Journal of Network and Computer Applications*, vol. 174, January 2021.
[8] J. A. Fraire, P. Madoery, S. Burleigh, M. Feldmann, J. Finochietto, A. Charif, N. Zergainoh, and R. Velazco, "Assessing contact graph routing performance and reliability in distributed satellite constellations," *Journal of Computer Networks and Communications*, July 2017.
[9] M. Moy, R. Kassouf-Short, N. Kortas, J. Cleveland, B. Tomko, D. Conricode, Y. Kirkpatrick, R. Cardona, B. Heller, and J. Curry, "Contact multigraph routing – overview and implementation," in *2023 IEEE Aerospace Conference*, 2023.
[10] S. Jain, K. Fall, and R. Patra, "Routing in a delay tolerant network," in *PROCEEDINGS OF ACM SIGCOMM*, 2004.
[11] O. De Jonckère and J. A. Fraire, "A shortest-path tree approach for routing in space networks," *China Communications*, vol. 17, no. 7, pp. 52–66, 2020.
[12] M. Fredman and R. Tarjan, "Fibonacci heaps and their uses in improved network optimization algorithms," in *25th Annual Symposium on Foundations of Computer Science, 1984*, 1984.
[13] NASA, "High-rate delay tolerant networking," 2023. [Online]. Available: https://github.com/nasa/HDTN

APPENDIX

Algorithms 4, 5 and 6 are the Contact Graph Dijkstra Search from [7], except line 7 of the Contact Selection Procedure (Algorithm 6), where the **if** condition has been changed from $C.arr\_time > BDT$ to $C.arr\_time \geq BDT$. This causes Algorithm 4 to terminate if the arrival time to the route's destination is less than or equal to all unvisited contacts, in which case an optimal path has been found.

Algorithms 7 and 8 give the modified versions of the algorithms, described in Section III-B, which are used below in the proof of Theorem 1. In both cases, the cleared working area of CP means that $C.visited = False$, $C.pred$ is empty, and $C.arr\_time = \infty$ for all contacts C, except $C_{root}.arr\_time$ is set to be the starting time of the expected route. Additionally, in the case of Algorithm 4, initially $C_{root}.visited\_n$ contains the source node S and all other $C.visited\_n$ are empty.

Algorithm 4 Contact Graph Dijkstra Search (Simplified)

```
Data: root contact $C_{root}$, destination $D$, contact plan $CP$
    (with cleared working area)
Result: Route $R_S^D$ from source $S$ to destination $D$
 1: $R_S^D = \{\}$
 2: $C_{fin} = \{\}$
 3: $BDT = \infty$
 4: $C_{curr} = C_{root}$
 5: while true do
 6:   $C_{fin}, BDT = \text{CRP}(CP, C_{curr}, C_{fin}, BDT)$
 7:   $C_{next} = \text{CSP}(CP, C_{curr}, BDT)$
 8:   if $C_{next} \neq \{\}$ then
 9:     $C_{curr} = C_{next}$
10:   else
11:     break
12:   end if
13: end while
14: route reconstruction using predecessors to find $R_S^D$
```

We now prove Theorem 1. The main idea comes from the behavior observed in the example in Section II-A that the contacts explored were in one-to-one correspondence once redundancies are removed. Algorithm 7 results from removing all such redundant steps, and we proceed to check that this algorithm essentially performs the same search as Algorithm 1.

Proof of Theorem 1. The data stored by a contact in Algorithm 7 correspond to the data stored by that contact's destination in Algorithm 1; for any $n \geq 1$, on the n^{th} iteration of the **while** loop of Algorithm 7, the destination of C_{curr} is the same as v_{curr} on the n^{th} iteration of the **while** loop of Algorithm 1, as guaranteed by the loop conditions.

Moreover, we show at the beginning of the n^{th} loops, the "visited" marker of any contact in Algorithm 7 is equal to that of the contact's destination in Algorithm 1, and that the arrival time of any vertex v in Algorithm 1 is equal to the minimum arrival time of all contacts with destination v in Algorithm 7.

Algorithm 5 Contact Review Procedure (CRP) (Simplified)

```
Data: $CP$, $C_{curr}$, $C_{fin}$, $BDT$
Result: revised $CP$, $C_{fin}$, $BDT$
 1: for contact $C \in CP$ with $C.src = C_{curr}.dst$ do
 2:   if $C.end \leq C_{curr}.arr\_time$ then
 3:     skip $C$
 4:   end if
 5:   if $C.visited$ then
 6:     skip $C$
 7:   end if
 8:   if $C.dst \in C_{curr}.visited\_n$ then
 9:     skip $C$
10:   end if
11:   $arr\_time = \max(C.start, C_{curr}.arr\_time) +$
      $C.owlt + owlt_{mgn}$
12:   if $arr\_time < C.arr\_time$ then
13:     $C.arr\_time = arr\_time$
14:     $C.pred = C_{curr}$
15:     $C.visited\_n = C_{curr}.visited\_n + C.dst$
16:     if $C.dst = D$ and $C.arr\_time < BDT$ then
17:       $BDT = C.arr\_time$
18:       $C_{fin} = C$
19:     end if
20:   end if
21: end for
22: $C_{curr}.visited = True$
```

Algorithm 6 Contact Selection Procedure (CSP) (Simplified)

```
Data: $CP$, BDT
Result: $C_{next}$
 1: $C_{next} = \{\}$
 2: $t_{earliest\_arrival} = \infty$
 3: for contact $C \in CP$ do
 4:   if $C.visited$ then
 5:     skip $C$
 6:   end if
 7:   if $C.arr\_time \geq BDT$ then
 8:     skip $C$
 9:   end if
10:   if $C.arr\_time < t_{earliest\_arrival}$ then
11:     $t_{earliest\_arrival} = C.arr\_time$
12:     $C_{next} = C$
13:   end if
14: end for
```

Similarly, BDT in Algorithm 7 corresponds to $v_d.arr\_time$ in Algorithm 1, and we will also note that C_{fin} corresponds to $v_d.pred$.

We use induction, tracking each of the values listed, and show that the algorithms terminate at the corresponding iterations. For the base case, at the beginning of the first iteration, the initial values meet the description above, with the arrival times of C_{root} and v_r equal to the initial time, all other arrival times equal to infinity, and all "visited" markers false. For the inductive step, we show that if the description holds at the beginning of an iteration of the **while** loops, then it

Algorithm 7 Modified Contact Graph Dijkstra Search

```
Data: root contact $C_{root}$, destination $D$, contact plan $CP$ (with cleared working area)
Result: Route $R_S^D$ from source $S$ to destination $D$
 1: $R_S^D = \{\}$
 2: $C_{fin} = \{\}$
 3: $BDT = \infty$
 4: $C_{curr} = C_{root}$
 5: while true do
 6:     $C_{fin}, BDT = \text{MCRP}(CP, C_{curr}, C_{fin}, BDT)$
 7:     $C_{next} = \text{CSP}(CP, C_{curr}, BDT)$
 8:     if $C_{next} \neq \{\}$ then
 9:         $C_{curr} = C_{next}$
10:     else
11:         break
12:     end if
13: end while
14: route reconstruction using predecessors to find $R_S^D$
```

Algorithm 8 Modified Contact Review Procedure (MCRP)

```
Data: $CP$, $C_{curr}$, $C_{fin}$, BDT
Result: revised $CP$, $C_{fin}$, $BDT$
 1: for contact $C \in CP$ such that $C.src = C_{curr}.dst$ do
 2:     if $C.end \leq C_{curr}.arr\_time$ then
 3:         skip $C$
 4:     end if
 5:     if $C.visited$ then
 6:         skip $C$
 7:     end if
 8:     $arr\_time = \max(C.start, C_{curr}.arr\_time) + C.owlt + owlt_{mgn}$
 9:     if $arr\_time < C.arr\_time$ then
10:         $C.arr\_time = arr\_time$
11:         $C.pred = C_{curr}$
12:         if $C.dst = D$ and $C.arr\_time < BDT$ then
13:             $BDT = C.arr\_time$
14:             $C_{fin} = C$
15:         end if
16:     end if
17: end for
18: for contact $C \in CP$ such that $C.dst == C_{curr}.dst$ do
19:     $C.visited = True$
20: end for
```

holds at the beginning of the following iteration. The **while** loops first apply Algorithms 2 and 8. By the assumption that the destination of C_{curr} is equal to v_{curr}, the first loops of Algorithms 2 and 8 search through the same set of contacts. The current contact C_{curr} always has the minimal arrival time of unvisited contacts, so by assumption it agrees with the arrival time of v_{curr}; thus, the first **if** statements skip the same contacts. Similarly, the second **if** statements skip the same contacts because the $C.visited$ in Algorithm 8 agrees with $C.dst.visited$ in Algorithm 2. The values of arr_time computed on line 8 of each algorithm agree, as $C_{curr}.arr\_time = v_{curr}.arr\_time$. At this point, the two algorithms update arrival times. In Algorithm 2, once the **for** loop has finished, the arrival time has been reviewed for each vertex v that was the destination of some contact of the loop. Its arrival time has either been left unchanged if it could not be improved, or it has been updated to the minimal arr_time found for contacts C in the loop such that $C.dst = v$. In Algorithm 8, the arrival times of the contacts are updated if they can be improved. The arrival times are not modified for the remainder of the **while** loops of Algorithms 1 and 7, so we have verified that at the beginning of the next loops, the arrival time of any vertex v in Algorithm 1 is equal to the minimum arrival time of all contacts with destination v in Algorithm 7. Predecessors are updated in both algorithms. The **if** statement on lines 12-15 of Algorithm 8 make sure that if BDT is updated, it is assigned the minimal $C.arr\_time$ amongst contacts C considered by the **for** loop with $C.dst = D$. This is exactly the value assigned to $v_d.arr\_time$, if it is updated, in the **for** loop of Algorithm 2, as required. By assumption (1) in the statement of the theorem, the orders of the contacts match between the two algorithms, so if C_{fin} is updated in Algorithm 8, it is assigned to be the same contact as $v_d.pred$ in Algorithm 2; this assumption is only needed in cases when two different contacts can realize the same arrival time to the same destination. Finally, at the end of Algorithm 8, all contacts with destination equal to $C_{curr}.dst$ are marked as visited, whereas in Algorithm 2, it is only v_{curr} that is marked as visited. This verifies that in the following iteration of the **while** loops, the "visited" marker of any contact in Algorithm 7 is equal to that of the contact's destination in Algorithm 1.

The next steps of the **while** loops in Algorithms 1 and 7 are the selection procedures, Algorithms 3 and 6. As shown above, if a contact has not yet been marked as visited in Algorithm 6, then its destination has not yet been marked as visited in Algorithm 3, and BDT corresponds to $v_d.arr\_time$. Thus, if a contact is not skipped in the **for** loop of Algorithm 6, its destination vertex is not skipped in the **for** loop of Algorithm 3. Since the arrival time at a vertex v in Algorithm 3 matches the minimal arrival time of contacts with destination v in Algorithm 6, the two **for** loops will find the same values of $t_{earliest\_arrival}$. Assumption (2) in the statement of the theorem is only necessary in the case when multiple vertices have the same arrival times in Algorithm 3: assuming compatible orders in the **for** loops, the final value of v_{next} in Algorithm 3 will be the destination of the final value of C_{next} in Algorithm 6. Returning to Algorithms 1 and 7, the final lines of the while loops update by setting $v_{curr} = v_{next}$ and $C_{curr} = C_{next}$ as long as v_{next} and C_{next} are not empty. By the above, this verifies that in the following **while** loops, the destination of C_{curr} in Algorithm 7 is equal to v_{curr} in Algorithm 1. The while loops terminate together if both v_{next} and C_{next} are empty. This completes the inductive step, and since we determined that the **while** loops terminate at their corresponding iterations, we conclude that Algorithms 1 and 7 require the same number of iterations.

$\square$