COLORADOSCHOOLOFMINES
EARTH • ENERGY • ENVIRONMENT

CSCI 370 Final Report

Sat Solvers (Davidson 1)

Jack Hall
Rygar Schyberg
Mallory Shaloy
Leif Wegener

Revised June 12, 2025

CSCI 370 Summer 2025

Mr. Scott Jensen

Table 1: Revision history

Revision	Date	Comments
New	5/12	Initial document created
Rev - 2	5/13	Minor Update to change team and member names
Rev - 3	5/14	Add barebones definition of done
Rev - 4	5/15	Write initial introduction, functional requirements, and definition of done
Rev - 5	5/17	Update risks
Rev - 6	5/18	Finished Initial Draft of Report for sections I-V
Rev - 7	5/21	Create system architecture and minor update to team profile
Rev - 8	5/22	Minor update to definition of done
Rev - 9	5/30	Start software test and quality, and ethical considerations
Rev - 10	6/1	Update software test and quality, and ethical considerations
Rev - 11	6/4	Expanded System Architecture section
Rev - 12	6/7	Minor Section title update
Rev - 13	6/8	Write future work, project results, and lessons learned sections
Rev - 14	6/9	Polish all sections, and update results and add graphs
Rev – 15	6/10	Polish all sections and update definitions
Rev – 16	6/12	Update throughput simulation test, and ethical considerations.

Table of Contents

Figure Table

I. Introduction

I.I High Level Scope

With the rapid expansion of satellite communications networks and low Earth orbit constellations, the importance of innovative solutions in the space domain software application realm is more important than ever. The purpose of this study is to tackle the challenge of optimally routing data through a dynamic network of satellites in real-time, with a focus on minimizing latency while also maximizing data throughput and maintaining reliability. The goal of this application is to contribute to more efficient inter-satellite communication, leading to more robust and performant global connectivity [1].

I.II Client

The client for this project is Davidson Technologies, Inc. (Davidson). They are a 29-year-old small business leading in digital engineering, cybersecurity, missile defense, and mission-enabling algorithms. They deliver solutions to the Department of Defense, with a focus on aerospace operations and contested environments.

I.III Definitions

LEO - Low Earth Orbit. Refers to satellites orbiting at less than 2,000 km above the surface of the Earth.
MEO - Medium (mid) Earth Orbit. Refers to satellites orbiting between 2,000 and 35,780 km.
Geosynchronous - A type of satellite orbit that follows the rotation of the Earth. A satellite in this orbit will appear to always have the same longitude from the ground. About 35,786 km above the surface of the Earth.
Geostationary - A type of Geosynchronous orbit above the equator. A satellite in this orbit will appear stationary from the ground.
TLE - Two/Three Line Element Set. Data format encoding information about a satellite.
SGP4 - Mathematical model for predicting the position and velocity of satellites based on a TLE
Satellite Constellation - A group of satellites with one shared purpose or control.
H3 - Discrete global grid system developed by Uber.
ISL - Intersatellite Links. Technology that enables communication between satellites in a constellation.

I.IV Stakeholders

The primary stakeholders for this project are Davidson and their clients. Davidson is looking to eventually launch satellites and expand into the space domain, and both Davidson and the DoD have an interest in the project. Furthermore, the U.S. government and its allies stand to benefit from a more robust and performant global communication network.

I.V Maintenance

Davidson is responsible for maintaining any software generated by this project. Davidson is considering using the results of the project in a future partnership with Mines Advanced Software Engineering students. In such a case, the responsibility of maintaining software may temporarily be transferred to the student-led team

II. Functional Requirements

The main components of this project are a simulation of a constellation of satellites, a routing algorithm for efficient satellite communications, and performance benchmarks.

II.I Simulation

A simulated constellation of satellites in low earth orbit in order to test our communications routing algorithm and compare it to existing ones. The simulation replicates orbital trajectories, inter-satellite communication variability, and communication channel characteristics. It is also designed to simulate satellite failure to test our algorithms' adaptability.

II.II Network representation

As part of our algorithm development, we create a dynamic graph with each vertex as a satellite. This graph simulates latency between satellites as well as accounting for bandwidth and error rates.

II.III Algorithm

The algorithms are intended to provide a way for satellites to communicate optimally. The algorithms are designed with the intention to minimize latency while maximizing throughput, while also adapting to dynamic networks. Additionally, they are designed to continue to operate efficiently when unexpected events occur, such as multiple satellite failures.

II.IV Performance Benchmarks

We evaluate our algorithm against traditional routing strategies using multiple test scenarios to highlight improvements in latency and throughput. We also simulate scenarios such as high network loads, satellite outages, and performance over time.

III. Non-Functional Requirements

III.I Speed

Both the algorithm and the simulation should run quickly, taking at most a second to fully simulate communications.

III.II Scalability

Both the algorithm and the simulation should be able to handle larger constellations of satellites with relative ease. The algorithm should still be performant with constellations of 500 satellites.

III.III Adaptability

Both the algorithm and the simulation should be able to adapt to any changes in the satellite constellation and react accordingly. This includes the addition or removal of nodes and edges as well as changes in the distances between satellites.

III.IV Research

All members of the team must have done prior research on the subject to fully understand the challenges of the problem. Additionally, team members must ensure that the approaches taken in this paper offer some form of novel approach which can be further researched.

IV. Risks

IV.I Technological Risks

The primary risk for the project is critical failure of the software. We need to ensure that the software is robust and does not crash, so satellites are not taken out of commission. We also need to ensure that there is nothing wrong with the algorithm that does not cause any network errors such as broadcast storms or infinitely looping routes. There is also the risk of insecure satellites or communications being intercepted. Additionally, there is the risk of issues with our simulation model leading to a suboptimal algorithm being chosen and us misinforming our client.

IV.II Developmental Risks

The primary developmental risks of this project are related to the specialized nature of the project. Team members need to be informed on both standard networking practices and the intricacies of satellite communication. Additionally, due to the program being written in rust, development may take longer than expected because the team is unfamiliar with the language.

V. Definition of Done

V.I Prototype Algorithm(s)

Produce a novel routing algorithm or a novel implementation of an existing routing algorithm. This algorithm is intended to be used on inter satellite communication and should maximize throughput and reliability while also minimizing latency to whatever extent possible.

V.II Software Prototype

Produce a software implementation of our designed algorithm which is capable of being run on a simplistic satellite constellation model. It additionally can be compared to other algorithms which predated the project and are results of the project.

V.III Benchmark Comparison Data

1. Create a viable satellite model
2. Test the proposed algorithms
3. Compare proposed algorithms to existing solutions and other proposed algorithms

V.IV Client Approval

1. Deliver technical white paper to the client which contains the results of the project and research which informed the project
2. Update repository with final version of satellite model
3. Update repository with final version of algorithm software
4. The client accepts the work

VI. System Architecture

The core of our system architecture is based on SGP4, a model for predicting the positions and velocities of satellites based on TLEs. With this we are able to calculate which satellites have a direct line of sight with each other at any given time, and how far apart they are from one another. We use this information to build our graphical network, with satellites as our nodes, line of sight for our edges, and latency for our edge weights. Our network model also simulates sending message packets, accounting for the load of each satellite. We utilize this representation to create and evaluate multiple algorithms for routing messages through satellite networks.

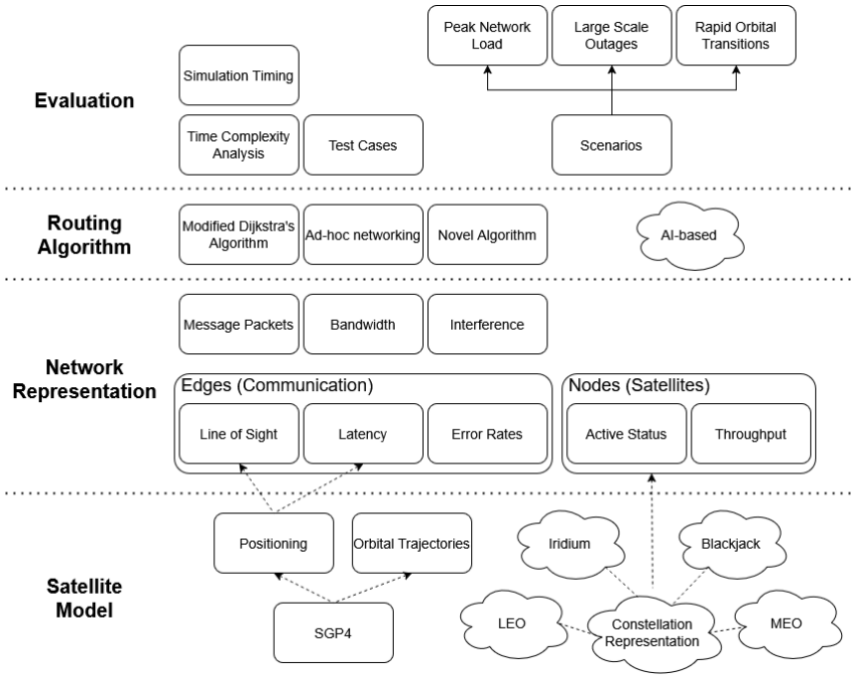

Figure 1: Project Architecture

The model starts by reading TLE (two-line element set) data, which contains information about the satellites being used for the simulation. These TLE files are then processed into a specific satellite structure which contains all relevant information about the satellite, such as their unique ID, TLE, load, etc. There are two separate versions of the model that are chosen depending on which algorithm is being tested: a centralized model and a distributed model. A centralized model uses a centralized routing algorithm, which has full knowledge of the entire network it is attempting to route through. Meanwhile, a distributed model uses a distributed routing algorithm which only has localized knowledge of the network, such as the current node's neighbors.

If the algorithm being tested is a centralized routing algorithm, then it will call upon the centralized model and take in more situational data, such as the data load capacity of satellites, and create the data packet. The model will run the routing algorithm starting with the initial satellite and find the path which it deems most efficient based on the design of the algorithm. Once this is complete, it will run the packet through the satellite path, leaving the possibility of packet loss/corruption or a satellite not being able to send said packet due to moving out of range or failing. Once the simulation is complete, the time it takes to run the model will be returned to the user and then the centralized model will reset and wait for the next simulated situation.

If the algorithm is based on a distributed algorithm, it will first prepare the satellites and packets in the same manner as the centralized algorithm. Once this is completed, it runs the simulation starting at the first node and will call upon the distributed algorithm. The algorithm will then calculate the next destination which the packet should be sent to, given the data available at the time it is called upon. This information will change as the simulation runs to reflect the dynamically changing locations of the satellites as they orbit. Once the best next satellite has been calculated, the packet will be sent to the next destination satellite and will then repeat that process until it reaches its overall destination satellite. Once this happens, it will return the time it took for the algorithm to run in that scenario and return to a non-simulated state, awaiting the inputs for the next simulated scenario much in the same way as the centralized model.

This information is summarized in the finite state automata in Figure 2.

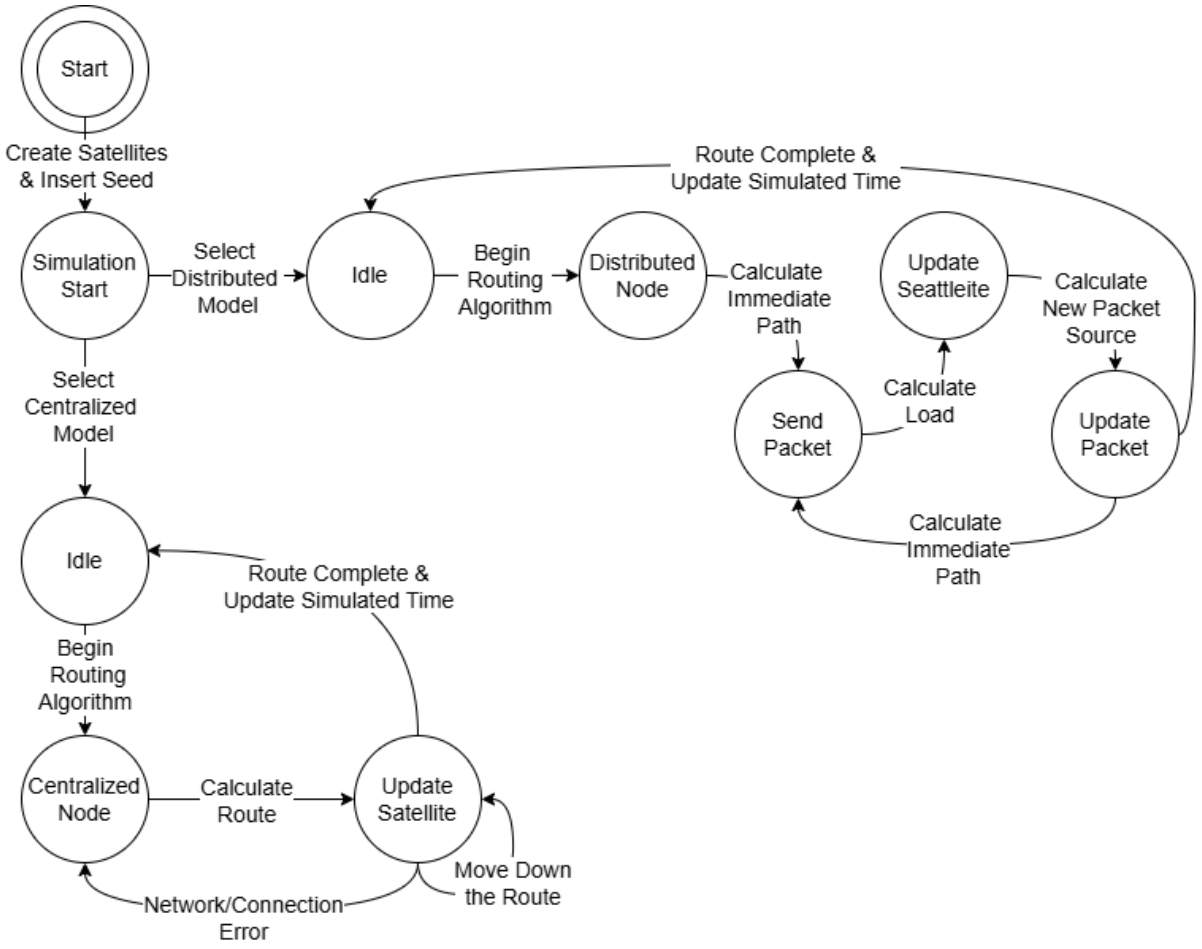

Figure 2: Finite State Automata for Satellite Model

VII. Algorithm Design

Given the problem, our primary design focus was the algorithms which we ran on the simulation model. The following algorithms are the ones which we designed or used as a benchmark comparison for the algorithms we designed. All algorithms were used or designed with the overall goal of finding the algorithm which would work best for a communications satellite system.

VII.I Greedy Search

A simple, best-effort distributed routing algorithm. The inspiration behind this algorithm was to implement a "distributed Dijkstra's Algorithm," where each satellite will determine the lowest cost path to its neighbors and send the packet to the most optimal neighbor. This process repeats with each node selecting the current best neighbor until the packet reaches its destination. To prevent looping, the packet contains information about which satellites have been visited, and which satellite sent the packet to which satellite (parents), allowing the algorithm to backtrack. This means that if the packet reaches a dead end, it can send it backwards down the route and explore other, less optimal paths in an attempt to stumble across the destination satellite.

This Greedy Search Algorithm determines edge weights by considering if the next satellite is closer to the destination than the source satellite (in other words, it will not try to send the packet away from the destination). After this, it will then consider the latency (immediate distance between the source satellite and its neighbor) and the load of the neighbor (how many other packets the neighbor is managing).

VII.II Ad-Hoc and Ad-Hoc with depth

The first version of this algorithm is a relatively basic ad-hoc algorithm implementation. When designing this algorithm, we made the assumption that a satellite would be able to obtain the location of both its neighbors and the destination point. This algorithm makes decisions based on which of the calculating satellite's neighbors is closest to the destination location. A packet may not be sent back to the satellite that transmitted to the current satellite. These choices were made because continually moving a packet towards the destination and removing transmission satellite ensured looping did not occur. It sends the data to the neighbor closest to the destination and repeats the process. This method also frequently results in fewer hops than other methods.

The second version of this algorithm takes the basic ad-hoc algorithm and adds logic for checking the neighbors of neighbors up to a specified depth with 1 being only immediate neighbors, 2 being immediate neighbors and those satellites immediate neighbors as well, etc. The investigation theorized that there might be a point at which the cost of pre-calculating neighbors would be a net benefit over time vs calculating at every node. Decisions on where to route a packet are still made on the distance to destination basis. Both versions of the algorithm also take satellite load into account, which can provide dynamic load balancing as well.

VII.III Centralized Point

The Centralized Point algorithm takes advantage of a specialized satellite network. This network has a set amount of MEOs that each manage a subnet of LEOs and act as a centralized node for routing in the subnet, using whatever centralized algorithm is most efficient. This algorithm has 4 scenarios, a LEO transmitting to its MEO, a MEO transmitting to another MEO, a MEO transmitting to the gateway LEO, and a LEO transmitting to another LEO in its subnet.

LEO to MEO: In this scenario, the LEO sends the packet directly to the MEO that manages its subnet.

MEO to MEO: Because the MEO network is small, we can use any distributed algorithm we want, such as greedy search or ad-hoc. In practice, the MEOs would have static routing tables as their orbits should be in such a way that their neighbors are consistent.

MEO to LEO: The algorithm would know the gateway LEO and use it as the packet source in a centralized algorithm like Dijkstra's. Once the route is calculated, it would directly send it to the gateway LEO. In our simulation, it dynamically chooses the closest MEO in the subnet to act as the gateway.

LEO to LEO: The simplest case where each LEO follows the route outlined by its managing MEO.

The Centralized Point algorithm allows us to distribute the intense computation load needed in centralized networking to many different satellites, each managing a small set of the entire network. This lowers the hardware requirements for routing while maintaining a centralized routing scheme.

VII.IV Dijkstra's Algorithm

Dijkstra's Algorithm is one of the most commonly used shortest path algorithms. It is currently the primary algorithm used in terrestrial networking. Because our network representation is an undirected weighted graph between satellites with edge weights determined by latency and network load, Dijkstra's algorithm is ideal. The main disadvantages of Dijkstra's algorithm are that it requires knowledge of the entire network and can be disoriented by the changing satellite system for longer transmission times. This algorithm assumes that our network has a high computational satellite that is able to manage the entire network and perform Dijkstra's Algorithm to route the entire network.

Our implementation of Dijkstra's algorithm first goes through each satellite in the network and checks each of its edges, keeping track of their most optimal neighbors. To reduce recompilations, it also keeps track of which satellites have been visited by the algorithm. Once complete, it will backtrack from the destination satellite back to its most optimal neighbor until it reaches the source, thus creating the route. This has a worst-case time complexity of O(n^2), n being the number of satellites. It also has a space complexity of O(n^2 + 3n), n^2 being the network representation and the 3n being the visited, parent, and delay vectors.

VII.V Grid Path

This algorithm was created as the result of attempting to create a method by which we could utilize a heuristic algorithm on a dynamic network. The team considered several different ways to convert the satellite network into a series of discrete, static networks. Both creating so-called network snapshots which would treat the satellite constellation as a static network for a set period and a grid-based system mapped to the surface of the Earth were considered for this purpose. Time did not permit exploration of every possible option, so we decided on the grid which is utilized in this algorithm. The Earth is subdivided into a set of hexagonal and pentagonal cells which are then used to calculate a gross path. From there, another algorithm is run to select the optimal path between satellites along the gross path. The current fine path algorithm employed is our implementation of Dijkstra's algorithm. Alternatively, the algorithm can weigh each satellite in each cell in the cell path, choosing the lowest weighted satellite in each cell to be the satellite used in the route. Our implementation used load as our weight.

This algorithm is built on the Rust implementation of Uber's open source H3. H3 is a discrete global grid system for indexing geographies into a hexagonal grid which allows latitude and longitude coordinates to be indexed to IDs that represent unique cells [2]. H3 offers a number of resolutions with increasing numbers of hexagonal cells. Our algorithm utilizes resolution 0 by default, which has 110 hexagonal cells and 12 pentagonal cells. To begin the calculation, the source satellite and the destination satellite are mapped to latitude and longitudes, then to H3 cells. From there, every satellite in the network is mapped into a cell and every cell that contains a satellite is considered to be valid. Valid cells then have their valid neighbors calculated. Once these calculations are complete, the heuristic algorithm a-star is applied to calculate the optimal path of valid cells between the beginning and ending cells. This generates a route of cells. A list of valid satellites along the cell route is generated and run through Dijkstra's algorithm, producing a final route. Our implementation is currently not well optimized, which could be an area of exploration for future work.

VIII. Software Test and Quality

VIII.I Software Quality Policies

During our software development for the network model, we have taken extra precautions to avoid any miscellaneous errors which might cause additional errors in the future. Our most basic quality assurance method is code reviews to make sure at least two members are aware of changes made to the main repository. Additionally, our code base is written in Rust, which holds a lot of restrictions on how memory can be used, allowing us to avoid all memory-related issues. Lastly, in regard to the algorithm implementations, all algorithms are predesigned/reviewed and put into a pseudocode format which is reviewed by the team before any implementation is attempted.

VIII.II Software Testing Model

The software testing model begins with creating a satellite constellation by selecting specific TLEs to input into our satellite model. This allows us to experiment with various orbits, such as polar orbits used in the Iridium constellation, the iridium constellation being a preexisting satellite network used as an example for research and testing.

The model also allows for selection of the routing algorithm to test as well as which model to test (distributed or centralized). Once the parameters are selected, the model runs a network simulation, recording relevant data such as the time it took for the message to reach the end satellite, the route which it traveled along, and the number of jumps required to reach the end destination.

We additionally have various scenarios to test the robustness of our algorithms and see how it redacts to various edge cases or extremes. Some of these scenarios were requested by our client, such as removing a critical node in the network, having an outage of random satellites, and experimenting with orbits. We have also produced our own test cases that were discussed in the previous section.

VIII.III Software Test Cases

We designed a number of test cases and test scenarios to analyze the performance of our algorithms and constellation arrangements. Test cases are problems we know the answers to, in order to test the correctness of our code. The test scenarios are designed to stress test our algorithms and may not have one correct answer. This is how we compare the performance of our algorithms.

Some examples of the test scenarios we designed were peak network load, where all satellites start near their max transmitting capacity. This test ensures that our algorithms account for load and allows us to compare how algorithms that use multicasting perform in high load conditions. We also have multiple scenarios to test satellite outages. One such test involves our algorithm needing to send packets away from the destination satellite initially due to breaks in the satellite network. Another involves a satellite along the optimal route going down after the first message is sent. These scenarios allow us to compare the performance of our algorithms and identify which satellites may perform well under some conditions, but falter under others.

VIII.IV Test Scenarios

VIII.IV.I Critical Node Removal

This test determines a critical node within our satellite network and removes it. This simulates the situation where a satellite becomes unavailable. Additionally, the test is used to track and show how well each algorithm routes around the removed satellite and how the new route and travel time changes.

VIII.IV.II Anti-Satellite Interceptor

This test is similar to the critical node removal test, where a satellite is removed from the network, except this test additionally removes some of the neighbors of the critical node. This simulates a scenario where an anti-satellite interceptor destroys a satellite, and the debris takes down neighboring satellites. The test analyses how our algorithms perform when a large number of satellites are out of commission.

VIII.IV.III Different Start Times

This test is used to determine how each algorithm reacts to our given sample satellite constellations given different start times. In other words, it tests how well our algorithms react to a variable network. This test tracks how the routes change and how much travel times change.

VIII.IV.IV Throughput/ Load Balancing

This test determines how well our algorithms avoid high-load satellites. It tracks how many packets an algorithm routes from a consistent start and end point before the algorithm attempts to route the packet to a satellite with a maximum load. It tracks the routes and travel times of each packet and how many packets it sent before sending to a maximum load satellite.

VIII.IV.V Ad-Hoc Depth Test

This test is only used on the ad-hoc algorithm. Because ad-hoc pre-calculates a path looking up to a specified number of neighbors away (depth), this test is used to determine the efficiency of different depths. It compares the packet's travel time, number of satellites it visited, and the routes the packets travel through, looking for any differences between different depths.

VIII.IV.VI Throughput Over Time Simulation Test

This test runs through a simulated network test for a specified duration. It tracks the number of packets attempted to be passed through the network and the number of packets that were successfully passed. In addition, it tracks the amount of time it takes for each packet to pass from start to finish, updating the network every 30 (simulated) seconds. This test runs through all currently implemented and functioning algorithms, using the grid route with Dijkstra. The data generated by this test can be considered an analog for raw algorithm-network throughput, with no load considerations.

VIII.IV.VII General Time Comparison

This is our most simplistic test yet one of the most important. We will allow each algorithm to run on a similar network with similar start and end satellites and find how much time it will take to transmit a specific number of packets. With this data, we then graph a plot of each algorithm based on the time and number of packets sent. We can use this data to confirm which algorithm runs the fastest in what scenarios.

VIII.V Final Report Quality Test

As a part of the requirements and our Definition of Done, we are expected to return a technical white paper to the client at the end of the project cycle. To ensure client approval and experience, this paper will be heavily reviewed before handing it over to the client. All members of the team will do a final review, and the paper will be compared with professional technical papers which will be procured through the Mines Library. Additionally, the team will attempt to have the paper reviewed by a third-party expert, the team's Colorado School of Mines advisor, who will be able to give an unbiased opinion while avoiding any conflict of interest on the part of Davidson Technologies.

IX. Project Ethical Considerations

IX.I Design Ethical Considerations

Since our project deals with potentially sensitive information, we inherently must consider how information is stored and sent in order to minimize the risk of leaking data. Especially in a project for a defense contractor, knowing that in the future this project could be used to send mission critical data, it was important for us to design with data safety in mind. While our project itself does not, we designed with the principle that data in a production environment would be encrypted. This is still a long way away from production, but our aim was to comply with sections 1.2 and 1.6 of the ACM Code of Ethics – to avoid harm and protect privacy [3]. When we simulate sending messages between satellites, we make sure to only send them when necessary. Data in a production environment should be encrypted, only sent when necessary, and not stored for longer than necessary.

IX.II Final Product Ethical Impact

Given that our client is a defense contractor, it is important to consider what the satellite program and the presumed future satellite network might be used for. While it could be useful for matters of national security, it's also worth noting that technology such as this has been used for more offensive purposes and can impact the lives of many people globally. Additionally, it is worth considering that through this project we are taking work from open-source projects — such as SGP4 and H3 — and using their work in a way which might be against the original intent of the developers. With our project we respect ACM 1.5 - to respect the work required to produce new ideas, inventions, creative works, and computing artifacts. We acknowledge and appreciate the work put into the open-source projects that enabled us to perform our work throughout the course of this field session. Finally, we believe our data analysis portion of this project complies with ACM 2.5 - to give comprehensive and thorough evaluations of computer systems and their impacts, including analysis of possible risks. To the best of our ability, we tried to give thorough evaluations of our developed algorithms and constellations, and to make the risks known.

X. Project Results

X.I Results

X.I.I Critical Node Removal

The primary indicator of performance is the least time loss from a satellite's removal. As expected, ad-hoc performed the best regarding consistency as the algorithm is designed to easily adapt to unknown situations with a time loss of 0.00001 to 0.0001 seconds. More interestingly however, is the performance of centralized point and Dijkstra's based grid algorithm. The centralized point algorithm had extremely minimal time loss within 0.00007 and 0.02 seconds depending on the size of the network. Similar numbers were shown by Dijkstra's grid-based algorithm. Both of which well out preform the adjustments made by other algorithms such as plain grid, greedy, and Dijkstra's.

X.I.II Anti-Satellite Interceptor

This test is similar to the critical node removal test, but analyses how our algorithms perform when a large number of satellites are taken out of commission, instead of just one. The performance of all our algorithms on this test from best to worst were Dijkstra's, grid with Dijkstra's, centralized point, grid choosing best, ad-hoc, and greedy. Dijkstra's was still able to find a good path even after the satellite interceptor. Both grid algorithms saw significant performance losses after the interceptor. Centralized point and greedy both saw similar runtimes, or even improvements after the interceptor, suggesting that they were not taking the optimal path initially. Ad-hoc saw very similar performance both before and after the satellites were taken out.

X.I.III Different Start Times

This test measures the reliance of the algorithms on specific conditions in a network at specific times. The best indicator of an algorithm's performance is consistency of time between the first and second run as well as the number of routes generated and used at each timestamp. Greedy, centralized point, and Dijkstra based grid all saw an increase in the number of routes from the first run time to the second run time, with the former seeing an increase of 11 routes and the latter two seeing an increase of 4 routes. Regarding time consistency, both grid-based algorithms and Dijkstra outperformed every other algorithm, primarily having differences in time less than 0.01 seconds.

X.I.IV Throughput/ Load Balancing

Load is handled in our model by allowing any given satellite to manage up to 100 packets at once. Once the satellite has 100 packets, it will be at maximum load and drop incoming packets. Additionally, the higher the load, the longer it will take the satellite to send the packet. Because of this, if a route passes packets through a specific satellite at any given time, then packets will get dropped as the satellite cannot handle it. This test

scenario tests how many packets an algorithm can route before losing a packet due to load, the minimum being 100 packets because of how our model was designed.

Ad-hoc and Centralized-point could only route 100 packets

- This is because, at the time of writing, these algorithms do not prioritize load, making routing decisions based on latency and distance only

Greedy search routed an average of 103 packets

- This is likely because greedy search looks at distance and immediate latency, and since the network is changing continuously, it can randomly send more than the minimum of 100 packets.
- This was done on both an 80-satellite network and a 529-satellite network

Base Grid search routed an average of 262 packets

- Grid search uses a-star on a grid network, with each cell in the grid having several satellites. A-star uses the average load of all the satellites in each cell as the weight, allowing it to route around high load satellites
- This was done on a 529 satellite network

Grid search with Dijkstra's had an average of 503 packets

- This uses the same grid network as mentioned above, with a-star routing the cells and Dijkstra's routing between the satellites in each cell
- This was done on a 529 satellite network

Dijkstra's algorithm had an average of 613 packets (80 satellite network) and 3640 (on 529 satellite network)

- By far the best algorithm for throughput as it determines routes based on both latency and load
- Dijkstra's algorithm accounts for the changing loads and updates each route accordingly, allowing it to support more packets without overloading any one satellite when compared to our other algorithms

X.I.V Ad-Hoc Depth Test

After comparing a variety of different ad-hoc depths ranging from 1 to 6 we determined that the depth has no appreciable effect on the route the ad-hoc algorithm creates. However, the depth does have a minor effect on the packet travel time, with larger depths having slightly shorter travel times (around 0.0002 – 0.0004 seconds faster). This is likely due to fewer satellites needing to calculate a route and instead simply sending the packet to the next hop.

X.I.VI Throughput Simulation Test

For the 80-satellite constellation simulation, each algorithm performed similarly. The exception to this is the grid path algorithm. This algorithm does not work consistently on a smaller network due to the implementation and after some time, the satellites progress to a state where there is no path between them on the grid due to some empty grid cells along the path. Outside of these, greedy was the worst performing algorithm, followed by ad-hoc/ad-hoc depth, and Dijkstra. On the 529-satellite constellation, again greedy had the worst performance, with ad-hoc and ad-hoc depth performing similarly. Dijkstra outperformed Grid-Dijkstra on this size network, but not significantly. The 600 second trial test on our three small to medium size constellations are summarized in Figure 3 below. It is likely that these results would change with a longer update time between calculations.

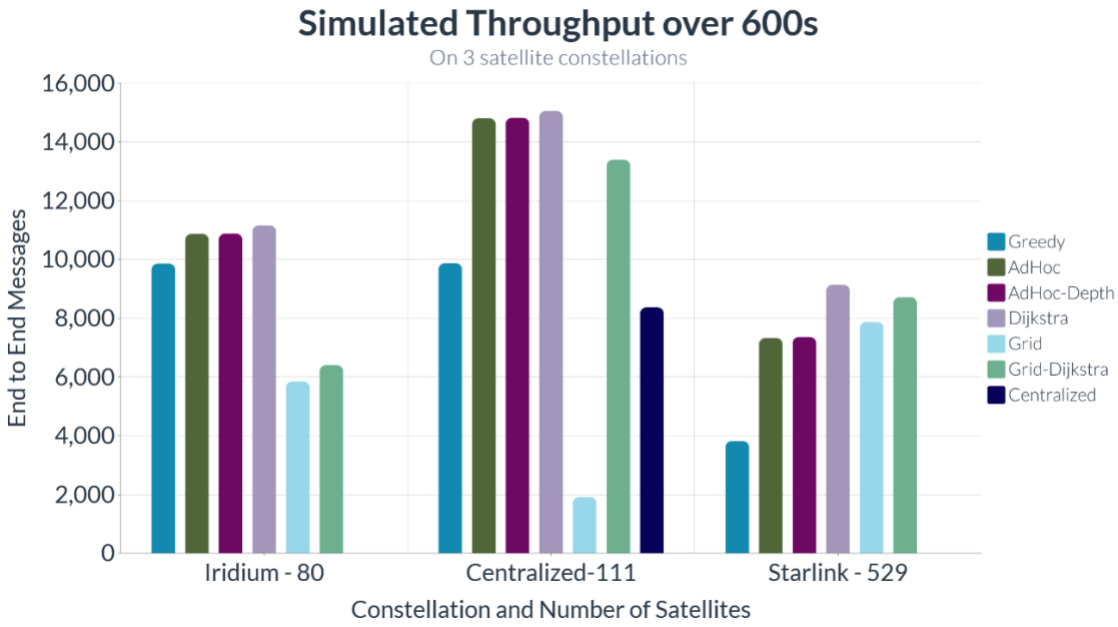

Figure 3: Results of 600s Simulated Throughput Test On 3 Small-Medium Satellite Constellations

On the 7608-satellite constellation, all the distributed network algorithms showed extremely poor performance. The Dijkstra's algorithm also began to show diminished performance with a network of this size. The grid-based algorithms both performed very well, with the Grid-Dijkstra version outperforming the straight grid until the 600 second duration trial when the grids were at resolution 0. When the grid resolution was increased to 2, the Grid-Dijkstra algorithm outperformed all others, while the grid only suffered a bit of a performance decrease. The results from this test are summarized below in Figure 4.

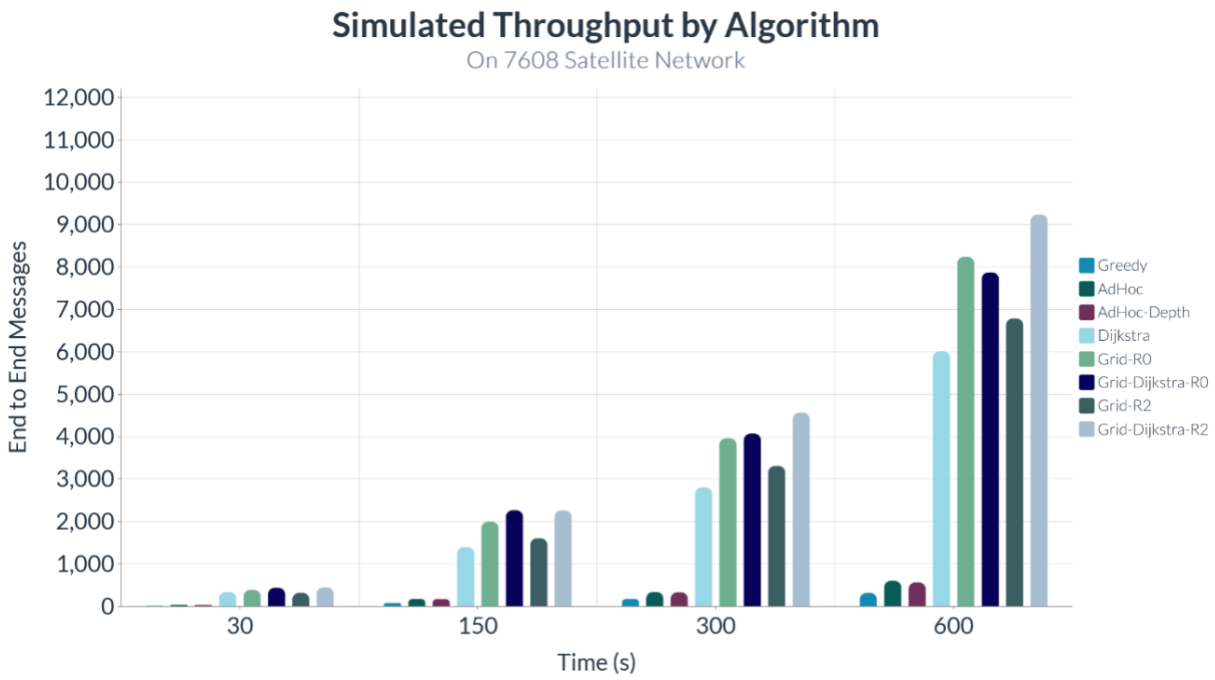

Figure 4: Results of Throughput Test on 7608 Satellite Constellation

X.I.VII General Time Comparison

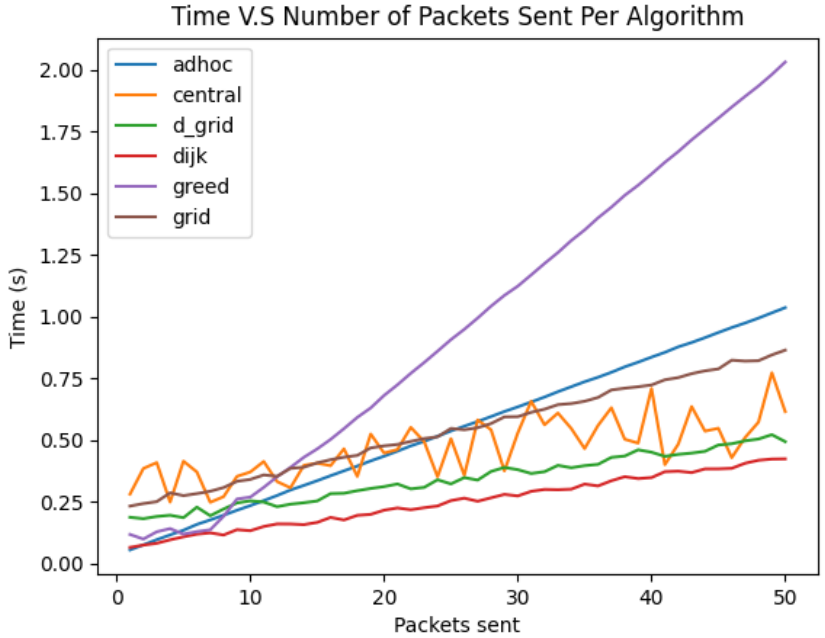

Figure 55: Graph of All Algorithms Comparing Packets Sent to Operation Time

The above graph shows the time to packet comparison for each algorithm. As expected, Dijkstra outperforms all other algorithms. But more interestingly, the Dijkstra based grid algorithm performed in a similar bracket to Dijkstra and maintained a lot of Dijkstra's stability in a relatively consistent amount of time needed even for larger numbers of packets. Notably, while centralized point behaved very erratically, at its best performance it outperformed the Dijkstra based grid algorithm.

X.II Best Overall Algorithm

Based on the results above, the team has determined that the best algorithm for satellite routing is the Dijkstra-based grid algorithm. While the current implementation of grid version does not have the same throughput on smaller constellations as standard Dijkstra's, it does reroute in singular satellite loss scenarios extremely efficiently only being outperformed by the ad-hoc algorithms which also have the drawbacks of lower throughput. Additionally, the algorithm was able to successfully adapt to different network conditions in the time test, which allowed for consistent operational time during varied network traffic conditions. Additionally, while the algorithm was not as efficient as standard Dijkstra's for rerouting when a larger portion of the network is nullified as in the anti-satellite interceptor test, it still outperformed all other algorithms. While the algorithm is slightly slower than Dijkstra's, it still outperformed all other algorithms and maintained the same levels of time constancy as the standard Dijkstra's. Finally, on a very large satellite constellation, it outperformed all other algorithms significantly in a throughput test. For these reasons, the team believes that this algorithm holds the best balance between speed, throughput, and reliability.

XI. Future Work

Since our project was so open-ended, there were a lot of different ways we could have done this project and still met the client's requirements. Our final project was just one approach to algorithmic improvement, and there is still work that could be done to optimize any of these solutions.

Firstly, while we created and experimented with several algorithms, our list is certainly not exhaustive. Potentially infinite algorithms for routing communications between satellites exist, and this is where we would look first for future work. More likely than not, there are algorithms better than any we tested. Furthermore, there is always room for more test scenarios. We implemented as many as we could within our timeline, but of course there are always more scenarios to be tested. Additionally, the centralized point algorithm test constellation was designed with a TLE combination of the global positioning system (GPS) and SpaceX's Starlink satellite constellation. A properly made TLE dataset which is specifically designed to match the centralized point algorithm would likely show better results. Additionally, creating a grid system which implements ad-hoc might show a significant improvement in the flexibility of the algorithm while maintaining data throughput. Our algorithms could also be further refined and optimized, especially grid routing.

Beyond algorithm development, our project could be improved in several ways. If there were a visualization of the satellite network, it would allow for quick visual checking of algorithm correctness and help facilitate demonstrations.

Finally, our simulation model could be further refined. As it currently stands, the simulation needs to be reset for different algorithms to be able to run. Ideally, the simulation could be controlled directly, changing the algorithm that is currently running and leaving the simulation itself alone. This would ensure more consistent comparisons across algorithms, and especially across different types of algorithms (centralized vs distributed). The simulation could also simulate imperfect information, and the delay that comes with updating what one satellite knows about satellites it doesn't have direct line of sight with. But primarily, the major revision which would assist the simulation is altering the operation of algorithms to more directly reflect the proper operational times. As the system currently stands, certain algorithms have extra overhead which might impact their

operational time. For example, the grid-based algorithms must recompute their grids during each run of the algorithm, which is an unrealistic addition as the grid-based algorithms would ideally be able to maintain a persistent image of the grid.

We are proud of the work that we put into this project, and what it is capable of currently, but nothing is perfect. With more time, there are many additional features that we would have implemented, and many we would have improved.

XII. Lessons Learned

In a project as broad as ours, there are plenty of opportunities to learn. Beyond the obvious lessons of how to work better in a team and specifically this team, there are a few major lessons we have learned from this project. The first is don't be afraid to pivot if something is not working the way you wanted or expected, even if you are fairly far down one path. Sometimes the only way to achieve a goal is to change direction, even when that means undoing a lot of work you have already done.

Another major lesson we've learned is that open-ended projects can be very good for learning; however, they can also be major sources of frustration. It can be helpful to add additional constraints onto a very open-ended project beyond what a client may give, at least first, so that you can quickly make some progress and determine if that is the direction you should pursue. To go along with this and tie into the previous lesson, be willing to admit when the direction chosen was incorrect, ideally sooner rather than later.

XIII. Acknowledgments

Firstly, we would like to thank Jameson Venema for bringing this project to the team and offering fantastic feedback and answers to our questions.

We would also like to thank Dr. Terry Bridgman of the Colorado School of Mines for all his help with mathematical assistance regarding our project and its problems as well as his suggestions for the ad-hoc and grid algorithms.

Additionally, we would like to thank our advisor Scott Jenson of the Colorado School of Mines for directional advice on proper development practices.

XIV. Team Profile

	Rygar Schyberg Computer Science - Robotics Background: Robotics, ARIA Lab, VEX Robotics Team Current Work: Programing robots
	Jack Hall Computer Science - General Background: Software Development, Robotics/Integrated Systems, Networking, Linux Environments Current Work: Teaching Assistant for Systems Programing/Computer Networks
	Mallory Shaloy Computer Science - Space Background: Software Development, Circuit Design, Systems Engineering Current Work: Personal Projects
	Leif Wegener Computer Science - General Background: Software Development, Networking, Cyber Defense Certificate Current Work: Personal Projects

References

[1] Davidson Technologies, Inc, [Online]. Available: https://cs-courses.mines.edu/csci370/FS2025S/Proposals/Davidson1.pdf. [Accessed 12 5 2025].

[2] Uber, "H3," Uber, [Online]. Available: https://h3geo.org/. [Accessed 9 6 2025].

[3] Association for Computing Machinery, "ACM Code of Ethics and Professional Conduct," [Online]. Available: https://www.acm.org/code-of-ethics. [Accessed 3 6 2025].

[4] T.-H. Chan, "A Localized Routing Scheme for LEO Satellite Networks," in *21st International Communications Satellite Systems Conference and Exhibit*, Yokohama, Japan, 2003.

[5] C. S. a. Y. Z. Tie Liu, "Load Balancing Routing Algorithm of Low-Orbit Communication Satellite Network Traffic Based on Machine Learning," *Wireless Communications & Mobile Computing (Online),* vol. 2021, 2021.

Appendix A - Key Terms

Include descriptions of technical terms, abbreviations and acronyms

Term	Definition
LEO	*Low Earth Orbit. Refers to satellites orbiting at less than 2,000 km in altitude*
MEO	*Medium (mid) Earth Orbit. Refers to satellites orbiting between 2,000 and 35,780 km in altitude*
Geosynchronous	*A type of satellite orbit that follows the rotation of the Earth. A satellite in this orbit will appear to always have the same longitude from the ground. About 35,786 km above the surface of the Earth*
Geostationary	*A type of Geosynchronous orbit above the equator. A satellite in this orbit will appear stationary from the ground*
TLE	*Two/Three Line Element Set. Data format encoding information about a satellite*
SGP4	*Open-source mathematical model for predicting the position and velocity of satellites based on a TLE*
H3	*Discrete global grid system developed by Uber*
ISL	*Intersatellite Links. Technology that enables communication between satellites in a constellation*