CPE 400 Final Project

Mesh Router Throughput Optimization With Loss Avoidance Using Modified A* Search Algorithm

Justin Howell

Elizabeth Kish

Martin Perez Navarro

Introduction

Problem Concept

The problem topic chosen is for the creation of a dynamic routing strategy that optimizes for throughput in a simulated mesh network. Throughput is a measurement of the amount of data that is sent from one destination and received at another within a timeframe measurement. The throughput of any given route is determined by a wide range of factors, including but not limited to the speed at which a packet can pass through a given path of routers, and the number of packets that are lost during transmission.

Proposed Project Solution

Our proposed solution is to utilize a modified A* search algorithm to navigate the mesh network, and find a path between two nodes, taking into account the “length” or time taken to traverse the path, as well as factoring in a given node’s likelihood to experience loss. The algorithm will take a generalized approach, assuming no prior information about the network or the number of packets to be sent through the network. By finding a path that is both short, and unlikely to experience loss, the searching strategy will be expected to maximize throughput on average for any number of packets that could be sent.

The A* search algorithm described can be utilized to search through a given graph, similar to Dijkstra's, but utilizes a heuristic value for each node in a graph. Typically this is an approximation of the distance to the final destination, which allows for A* to search through the graph with greater efficiency, however we will instead replace the heuristic value with an approximation of the likelihood or risk that a given node will experience loss. Modification was then further made to the algorithm, and the travel distance or shortest path distance is updated during run time to include a weighted factor of the heuristic value, so that the algorithm takes into account the risk of loss when determining the shortest path. A* is chosen as the search algorithm is more efficient, prioritizing searching paths first less likely to incur loss.

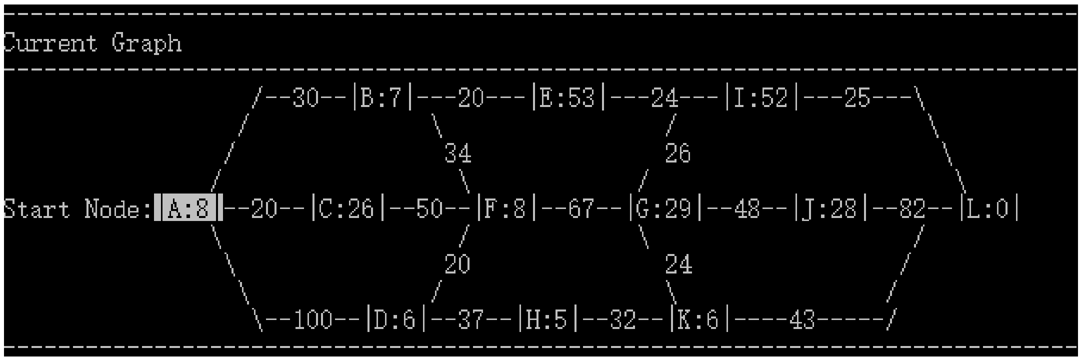
Functionality

Simulation Characteristics

The mesh network is represented utilizing an undirected graph, with each node of the graph representing a router, and containing the nodal processing delay, nodal transmission delay, and remaining buffer size before buffer overflow and loss occurs. The connections between each node are the data connections between the routers, and contain the propagation delay for data to travel from one end to the other. For user readability and faster calculation we utilize basic integers, scaled from ten to ninety depending on generated value.

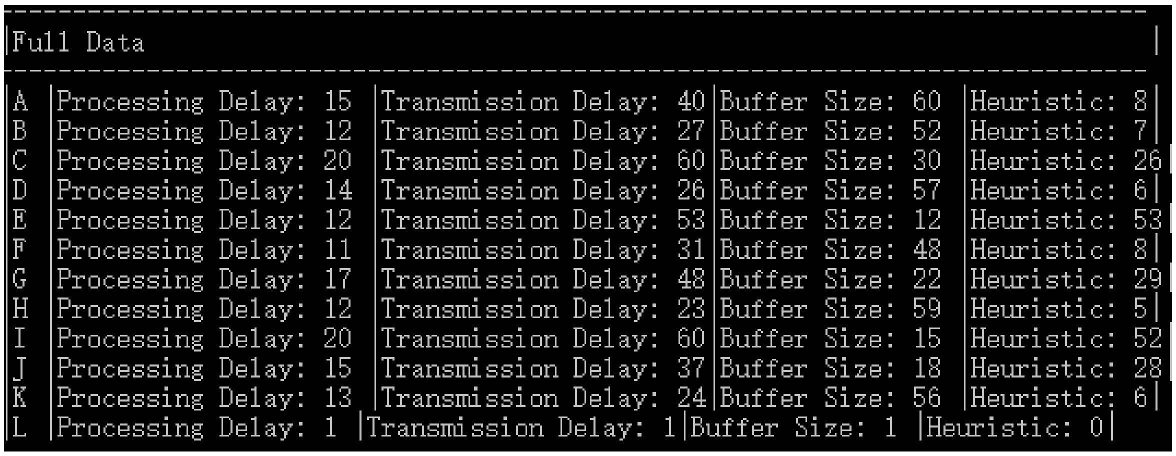
As stated previously, the algorithm assumes no prior information about the mesh network. As such, we are unable to generate a queuing delay value for the routers, as queuing delay relies on prior information on the average length of the queue for a given node. The code does pre-generate a graph and calculate it's heuristic values prior to running the algorithm, though this is only done for ease of access and use by the user. By pre-generating the graph, we can view a table of all generated values of the graph without running A* and was helpful in developing the project. This can be modified to calculate the heuristic during algorithm run-time, receiving the nodal delay data from each node and calculating the heuristic during node exploration.

Our code creates a twelve node graph to represent the mesh network and preloads a default graph to use. The default graph and the values of each node are shown below.



```
---------------------------------------------------------------------
Current Graph
---------------------------------------------------------------------
                 /--30--|B:7|---20---|E:53|---24---|I:52|---25---\
                /            \                /                   \
               /              34             26                    \
              /                \            /                       \
Start Node:|A:8|--20--|C:26|--50--|F:8|--67--|G:29|--48--|J:28|--82--|L:0|
              \                /            \                       /
               \              20             24                    /
                \            /                \                   /
                 \--100--|D:6|--37--|H:5|--32--|K:6|----43-----/
---------------------------------------------------------------------
```

Figure 1.0: Graph of the 12 node network, and the connection "length" values.



```
---------------------------------------------------------------------------------
|Full Data                                                                       |
---------------------------------------------------------------------------------
|A |Processing Delay: 15 |Transmission Delay: 40|Buffer Size: 60 |Heuristic: 8|
|B |Processing Delay: 12 |Transmission Delay: 27|Buffer Size: 52 |Heuristic: 7|
|C |Processing Delay: 20 |Transmission Delay: 60|Buffer Size: 30 |Heuristic: 26|
|D |Processing Delay: 14 |Transmission Delay: 26|Buffer Size: 57 |Heuristic: 6|
|E |Processing Delay: 12 |Transmission Delay: 53|Buffer Size: 12 |Heuristic: 53|
|F |Processing Delay: 11 |Transmission Delay: 31|Buffer Size: 48 |Heuristic: 8|
|G |Processing Delay: 17 |Transmission Delay: 48|Buffer Size: 22 |Heuristic: 29|
|H |Processing Delay: 12 |Transmission Delay: 23|Buffer Size: 59 |Heuristic: 5|
|I |Processing Delay: 20 |Transmission Delay: 60|Buffer Size: 15 |Heuristic: 52|
|J |Processing Delay: 15 |Transmission Delay: 37|Buffer Size: 18 |Heuristic: 28|
|K |Processing Delay: 13 |Transmission Delay: 24|Buffer Size: 56 |Heuristic: 6|
|L |Processing Delay: 1 |Transmission Delay: 1|Buffer Size: 1 |Heuristic: 0|
```

Figure 1.1: Table of each node and it's delay values, and the resulting heuristic.

Heuristic and Loss Avoidance Risk Calculation

We calculate the risk of loss utilizing the following equation:

$$weight * [(\frac{s_1}{buffer_{remaining}}) * (s_2 * (nodal_{processing} + nodal_{transmission}))]$$

The equation takes the inverse of the amount of space remaining in the buffer, and is then scaled by the time each node/router requires to place a packet on the next connection. We use this equation as a risk assessment of loss. The smaller the remaining buffer size is, the faster loss will occur if loss occurs in the given node. The longer it takes for a node to process and place a packet from it's buffer onto the connection, the higher the chances packet loss will occur.

We use scaling factors on each side of the equation, s_1 and s_2, to scale each side of the equation to be within a similar value range of the connection, so that the heuristics will be correctly factored into the A* search. When applying the heuristic to the shortest path, we also scale it by a weighting factor, or an approximation of how much the searching strategy will value the risk of loss.

The default values utilized to scale the various components of the heuristic is:

s_1: 1000 | s_2: .01 | weight: 1.1

Code Documentation

The code for the project is structured using object oriented programming. Each component of the graph is an object of one of three classes, Graph, Graph_Node, or Graph_Connection. At run-time, a graph with the default mesh network structure is created by creating each Graph_Node, which stores a vector of its Graph_Connection's, and are all stored in a Graph object.

The Graph object is then passed into a utility class named UI, which acts as the command line interface, and contains the modified A* algorithm. The UI class carries out one of five possible menu commands, and waits on the user to select which one. On input of one, the UI will display the current graph's connections and heuristic values. On an input of two, the UI will randomly generate new values for all nodes and connections, then display them for the user to see. Input three displays the full nodal delays for all current nodes. Inputs four and five execute the A* search algorithm, though at different speeds. The step-by-step A* will wait for the user to input one to continue to the next step, displaying the current graph and A* table values for the step, or allows the user to exit and return the main menu. An input of five will execute A* through the whole graph, outputting each step and the final path the algorithm found at the end, and returns to the main menu.

The A* code saves each node in a STL map for unvisited nodes, and moves them during exploration to a visited map. We utilize maps for this so the code can quickly access a targeted node regardless of the order of the map using simple ID value keys, without iterating through an array or a vector for the target node.

Pseudo Code for Modified A:*

```
WHILE current_node IS NOT final_node
    FOR first_connection_of_current_node TO final_connection_of_current_node
        Delay = connection_delay + (weight * target_node_heuristic)
        New_shortest = target_node_short + delay

        IF new_shortest < old_shortest
            target_node_short = new_shortset
            target_node_total = new_shortest + target_node_heuristic
            target_node_previous_node = current_node
            IF current_node IS IN UNVISITED MAP
                REMOVE current_node FROM UNVISITED MAP
                ADD current_node TO VISITED MAP
            ELSE
                REMOVE current_node FROM VISITED MAP
                ADD current_node TO UVISITED MAP
    ENDFOR

    FOR all_nodes_in_unvisited_map
        IF current_node_total_distance < smallest_total_distance
            smallest = current_node
    ENDFOR

    current_node = smallest
ENDWHILE
```

Error Handling

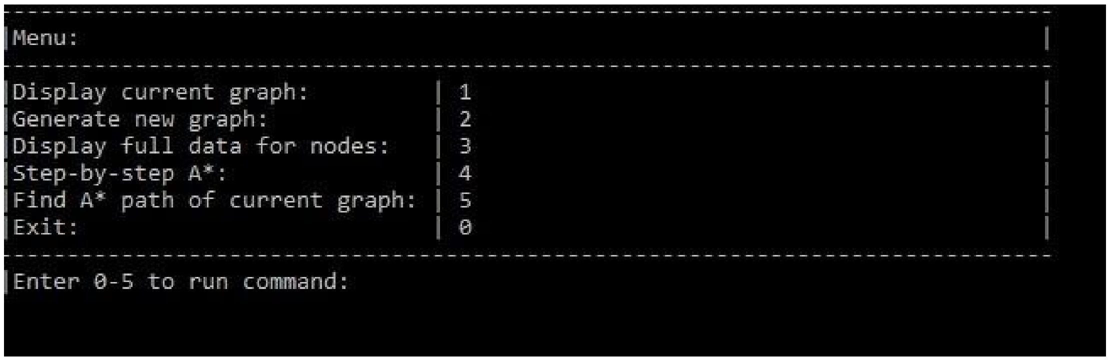
The code primarily uses objects and pointer referencing, so the two A* functions uses simple If-Else statement testing before each pointer is called to check if the pointer is or is not NULL. If it is not NULL, the code executes, and if the pointer is NULL then the code returns and prints the error code and corresponding pointer that was attempted to be called to the command line if possible, and exits the program or returns to the main menu.

Novel Contribution

The novel contributions made in this project include a generalized searching strategy that can be utilized at any point during a network's lifetime, assessing on exploration the various points of data to path find through the mesh. The novelty of the searching strategy is in the optimization of throughput while taking into account loss, as the algorithm will select "shorter" paths that are less likely to incur loss, or the loss they will experience to be as little as possible for as wide a range of number of data packets being transported through the network.

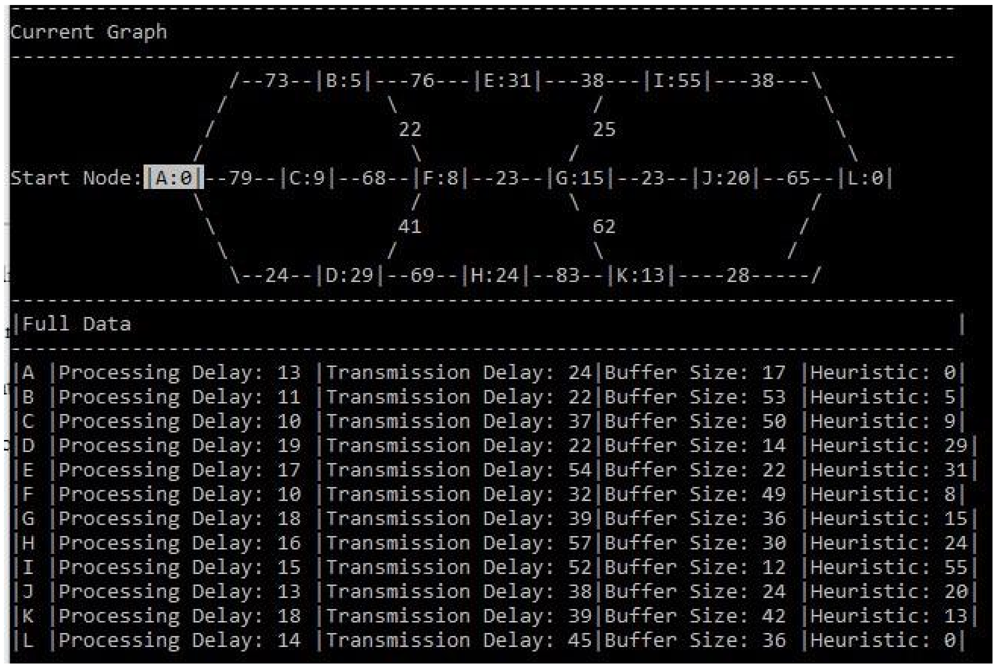
Another novel contribution is in the form of a modifiable function to calculate the heuristic. By including scaling values, the function allows for the application to be used with any range of values for the nodal processing, transmission, and propagation delay. We also include a weighting factor that allows for adjustable loss avoidance, as the value of the weighting factor determines how much of approximate risk value is applied to the shortest path distance. The utilization of heuristic values also allows for the code to run more efficiently, as the algorithm is better able to prioritize searching paths with low heuristic values.

Finally, we developed an interactive command line interface that allows for the user to visualize the A* algorithm as it runs by displaying both the A* value table and the current state of the traversal through the graph, as well as allowing the user to generate through one command new graphs for analysis.



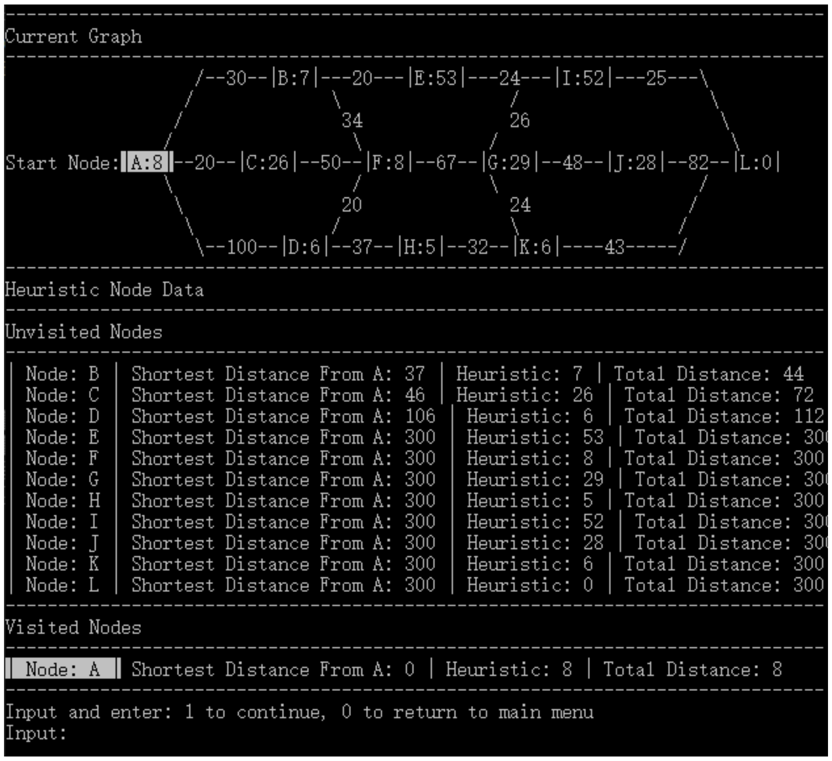
```
------------------------------------------------------------------------------
|Menu:                                                                       |
------------------------------------------------------------------------------
|Display current graph:          | 1                                         |
|Generate new graph:             | 2                                         |
|Display full data for nodes:    | 3                                         |
|Step-by-step A*:                | 4                                         |
|Find A* path of current graph:  | 5                                         |
|Exit:                           | 0                                         |
------------------------------------------------------------------------------
|Enter 0-5 to run command:
```

Figure 2.0: Main menu for project code.



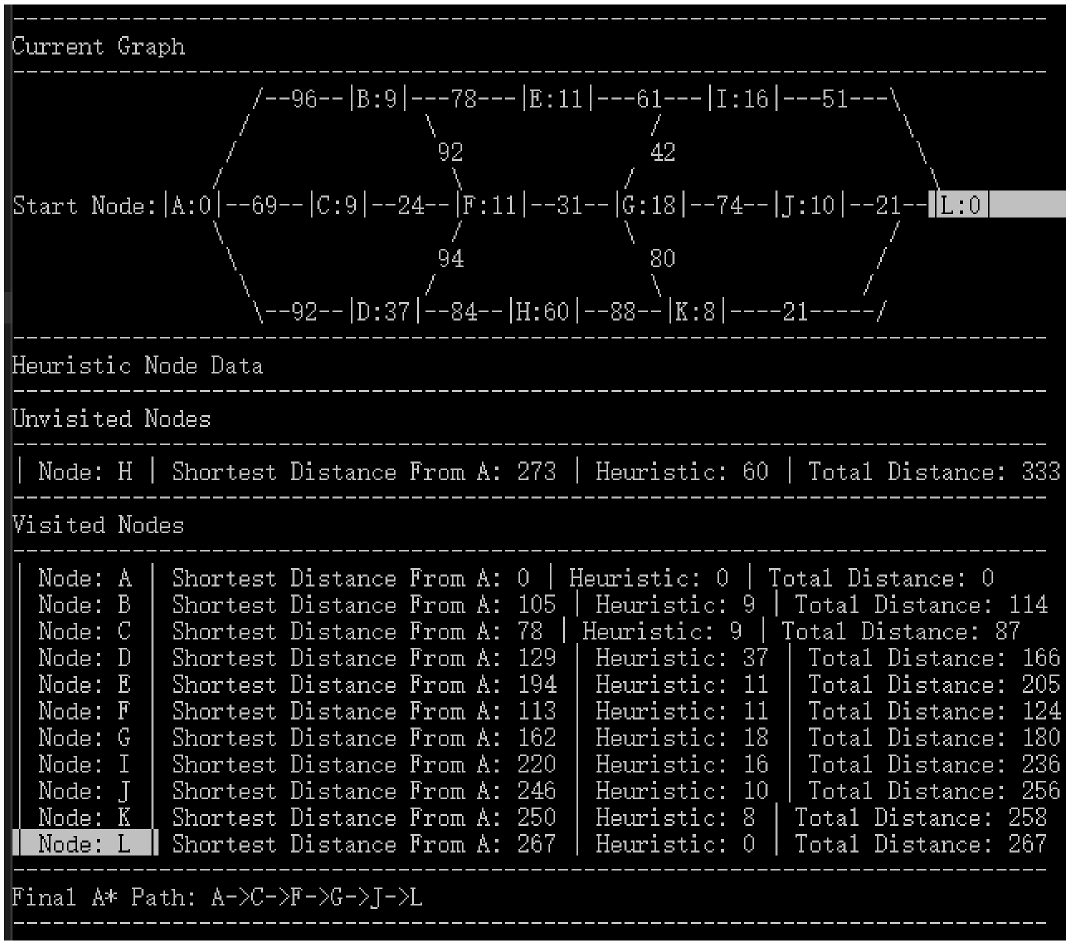
```
------------------------------------------------------------------------------
Current Graph
------------------------------------------------------------------------------
                  /--73--|B:5|---76---|E:31|---38---|I:55|---38---\
                 /             \                /                  \
                /               22              25                  \
               /                 \            /                      \
Start Node:|A:0|--79--|C:9|--68--|F:8|--23--|G:15|--23--|J:20|--65--|L:0|
               \                 /            \                   /
                \               41              62               /
                 \             /                \               /
                  \--24--|D:29|--69--|H:24|--83--|K:13|----28-----/
------------------------------------------------------------------------------
|Full Data                                                                   |
------------------------------------------------------------------------------
|A |Processing Delay: 13 |Transmission Delay: 24|Buffer Size: 17 |Heuristic: 0|
|B |Processing Delay: 11 |Transmission Delay: 22|Buffer Size: 53 |Heuristic: 5|
|C |Processing Delay: 10 |Transmission Delay: 37|Buffer Size: 50 |Heuristic: 9|
|D |Processing Delay: 19 |Transmission Delay: 22|Buffer Size: 14 |Heuristic: 29|
|E |Processing Delay: 17 |Transmission Delay: 54|Buffer Size: 22 |Heuristic: 31|
|F |Processing Delay: 10 |Transmission Delay: 32|Buffer Size: 49 |Heuristic: 8|
|G |Processing Delay: 18 |Transmission Delay: 39|Buffer Size: 36 |Heuristic: 15|
|H |Processing Delay: 16 |Transmission Delay: 57|Buffer Size: 30 |Heuristic: 24|
|I |Processing Delay: 15 |Transmission Delay: 52|Buffer Size: 12 |Heuristic: 55|
|J |Processing Delay: 13 |Transmission Delay: 38|Buffer Size: 24 |Heuristic: 20|
|K |Processing Delay: 18 |Transmission Delay: 39|Buffer Size: 42 |Heuristic: 13|
|L |Processing Delay: 14 |Transmission Delay: 45|Buffer Size: 36 |Heuristic: 0|
```

Figure 2.1: Example of command 2, displaying the graph's new randomly generated values and calculated heuristics.



```
------------------------------------------------------------------------------
Current Graph
------------------------------------------------------------------------------
                  /--30--|B:7|---20---|E:53|---24---|I:52|---25---\
                 /            \                 /                  \
                /              34               26                  \
               /                 \            /                      \
Start Node:|A:8|--20--|C:26|--50--|F:8|--67--|G:29|--48--|J:28|--82--|L:0|
               \                 /            \                    /
                \              20               24                /
                 \            /                 \                /
                  \--100--|D:6|--37--|H:5|--32--|K:6|----43-----/
------------------------------------------------------------------------------
Heuristic Node Data
------------------------------------------------------------------------------
Unvisited Nodes
------------------------------------------------------------------------------
| Node: B | Shortest Distance From A: 37 | Heuristic: 7 | Total Distance: 44
  Node: C | Shortest Distance From A: 46 | Heuristic: 26 | Total Distance: 72
  Node: D | Shortest Distance From A: 106 | Heuristic: 6 | Total Distance: 112
  Node: E | Shortest Distance From A: 300 | Heuristic: 53 | Total Distance: 30
  Node: F | Shortest Distance From A: 300 | Heuristic: 8 | Total Distance: 300
  Node: G | Shortest Distance From A: 300 | Heuristic: 29 | Total Distance: 30
  Node: H | Shortest Distance From A: 300 | Heuristic: 5 | Total Distance: 300
  Node: I | Shortest Distance From A: 300 | Heuristic: 52 | Total Distance: 30
  Node: J | Shortest Distance From A: 300 | Heuristic: 28 | Total Distance: 30
  Node: K | Shortest Distance From A: 300 | Heuristic: 6 | Total Distance: 300
| Node: L | Shortest Distance From A: 300 | Heuristic: 0 | Total Distance: 300
------------------------------------------------------------------------------
Visited Nodes
------------------------------------------------------------------------------
| Node: A | Shortest Distance From A: 0 | Heuristic: 8 | Total Distance: 8
------------------------------------------------------------------------------
Input and enter: 1 to continue, 0 to return to main menu
Input:
```

Figure 2.2: Example of command 4, with the program waiting on the user input before carrying out the next step of the A search.*
]

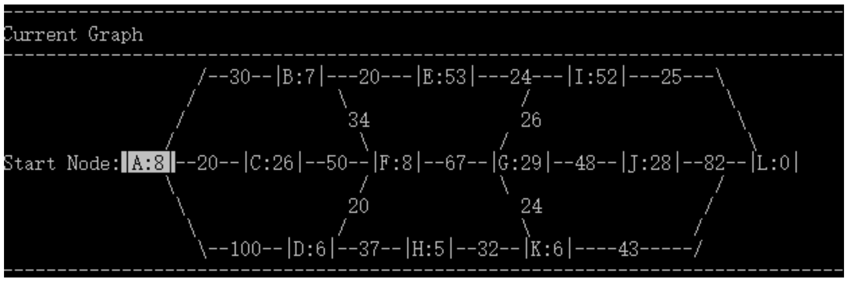


```
------------------------------------------------------------------------------
Current Graph
------------------------------------------------------------------------------
                  /--96--|B:9|---78---|E:11|---61---|I:16|---51---\
                 /             \                /                  \
                /               92              42                  \
               /                 \            /                      \
Start Node:|A:0|--69--|C:9|--24--|F:11|--31--|G:18|--74--|J:10|--21--|L:0|
               \                 /            \                     /
                \               94              80                 /
                 \             /                \                 /
                  \--92--|D:37|--84--|H:60|--88--|K:8|----21-----/
------------------------------------------------------------------------------
Heuristic Node Data
------------------------------------------------------------------------------
Unvisited Nodes
------------------------------------------------------------------------------
| Node: H | Shortest Distance From A: 273 | Heuristic: 60 | Total Distance: 333
------------------------------------------------------------------------------
Visited Nodes
------------------------------------------------------------------------------
| Node: A | Shortest Distance From A: 0 | Heuristic: 0 | Total Distance: 0
| Node: B | Shortest Distance From A: 105 | Heuristic: 9 | Total Distance: 114
| Node: C | Shortest Distance From A: 78 | Heuristic: 9 | Total Distance: 87
| Node: D | Shortest Distance From A: 129 | Heuristic: 37 | Total Distance: 166
| Node: E | Shortest Distance From A: 194 | Heuristic: 11 | Total Distance: 205
| Node: F | Shortest Distance From A: 113 | Heuristic: 11 | Total Distance: 124
| Node: G | Shortest Distance From A: 162 | Heuristic: 18 | Total Distance: 180
| Node: I | Shortest Distance From A: 220 | Heuristic: 16 | Total Distance: 236
| Node: J | Shortest Distance From A: 246 | Heuristic: 10 | Total Distance: 256
| Node: K | Shortest Distance From A: 250 | Heuristic: 8 | Total Distance: 258
| Node: L | Shortest Distance From A: 267 | Heuristic: 0 | Total Distance: 267
------------------------------------------------------------------------------
Final A* Path: A->C->F->G->J->L
------------------------------------------------------------------------------
```

Figure 2.3: Example of command 5, with the display of the final step of A after generating a new graph, the unvisited and visited node values, and the final "shortest" path value.*

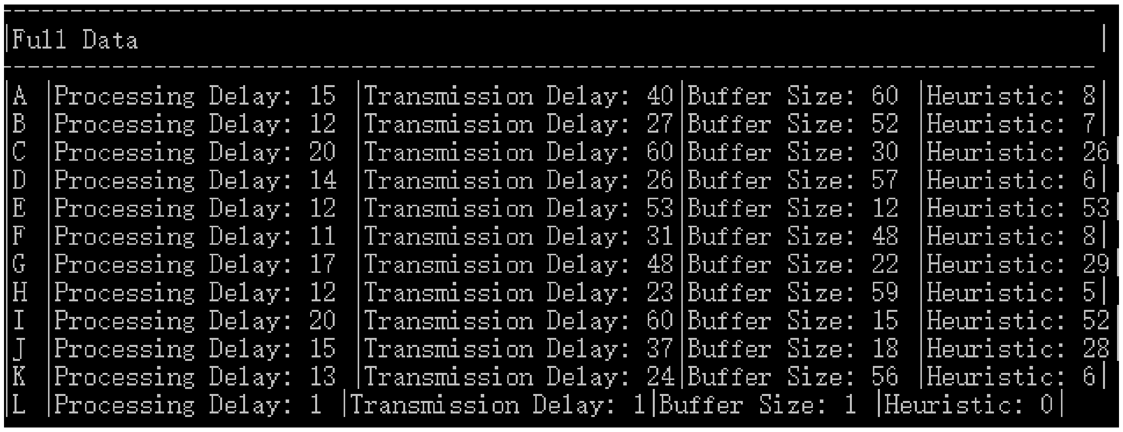
Results and Analysis

We will perform our analysis on a default graph that we generated, with purposefully planned extreme scenario nodes to showcase the throughput optimization versus a simple shortest path searching strategy.



```
------------------------------------------------------------------------------
Current Graph
------------------------------------------------------------------------------
                  /--30--|B:7|---20---|E:53|---24---|I:52|---25---\
                 /             \                /                  \
                /               34              26                  \
               /                 \            /                      \
Start Node:|A:8|--20--|C:26|--50--|F:8|--67--|G:29|--48--|J:28|--82--|L:0|
               \                 /            \                    /
                \               20              24                /
                 \             /                \                /
                  \--100--|D:6|--37--|H:5|--32--|K:6|----43-----/
------------------------------------------------------------------------------
```

Figure 3.0: Default graph used for analysis.



```
-------------------------------------------------------------------------
|Full Data                                                              |
-------------------------------------------------------------------------
|A |Processing Delay: 15 |Transmission Delay: 40|Buffer Size: 60 |Heuristic: 8|
|B |Processing Delay: 12 |Transmission Delay: 27|Buffer Size: 52 |Heuristic: 7|
|C |Processing Delay: 20 |Transmission Delay: 60|Buffer Size: 30 |Heuristic: 26|
|D |Processing Delay: 14 |Transmission Delay: 26|Buffer Size: 57 |Heuristic: 6|
|E |Processing Delay: 12 |Transmission Delay: 53|Buffer Size: 12 |Heuristic: 53|
|F |Processing Delay: 11 |Transmission Delay: 31|Buffer Size: 48 |Heuristic: 8|
|G |Processing Delay: 17 |Transmission Delay: 48|Buffer Size: 22 |Heuristic: 29|
|H |Processing Delay: 12 |Transmission Delay: 23|Buffer Size: 59 |Heuristic: 5|
|I |Processing Delay: 20 |Transmission Delay: 60|Buffer Size: 15 |Heuristic: 52|
|J |Processing Delay: 15 |Transmission Delay: 37|Buffer Size: 18 |Heuristic: 28|
|K |Processing Delay: 13 |Transmission Delay: 24|Buffer Size: 56 |Heuristic: 6|
|L |Processing Delay: 1 |Transmission Delay: 1|Buffer Size: 1 |Heuristic: 0|
```

Figure 3.1: Default graph values used for analysis.

The shortest path without accounting for loss can be calculated by manually performing Dijkstra's, and adding the connection delay and the nodal processing and transmission of each node for the path length. The shortest path is found to be **(A->B->E->I)** with a path length of 338**.** The path found by A* for the default graph is **(A->B->F->D->H->K),** with a path length of 444.

This is expected, as the shortest path was generated to have fewer number of steps, shorter propagation delays, but very high values for the transmission and processing delay, and very little buffer space remaining. On the other hand, the path found by A* has more steps, a longer path length, but low heuristic values for delay and large amounts of buffer space remaining. In the next section, we will see how this affects the throughput and loss.

We can analyze the throughput without accounting for loss for a given path by using the length of the path as the time taken for a given packet to travel from Node A to Node L. For the shortest path, that would be 1 packet per 338 time units, and for the A* path 1 packet per 444 time units. In this case, the shortest path has a higher throughput than the A* path.

Factoring in Loss

We can now factor in loss for the shortest path. We can identify whether or not a node in our simplified mesh network will or will not experience loss by analyzing the rate of incoming packets versus the rate of outgoing packets. A node's outgoing transmission rate is determined by the nodal processing delay plus the nodal transmission delay, or the time it takes a router to process and place an incoming packet on the wire out and out of the buffer. A node's incoming packet rate is then simply the outgoing rate of the node previous to it, as the rate at which the prior node is able to send the packets is the rate at which the current node will receive them.

Loss can then be identified to occur in nodes where the outgoing rate is greater than the incoming rate. When packets arrive at a rate faster than the node can remove them from the buffer, the buffer will begin to fill. The amount of lost packets can be determined using the following function.

$$L = B - \left(NumPackets_{node} - \frac{NumPackets_{node} * IncomingRate_{node}}{OutgoingRate_{node}}\right)$$

We examine the remaining buffer size of the given node, subtracting the number of packets that we will be sending into the node. This subtracts the number of packets multiplied by the incoming rate, divided by the outgoing rate to establish the number of packets that will be lost by the node.

We can look through the shortest path and the modified A* paths, and identify the nodes at which loss will occur by seeing which nodes have a larger outgoing rate in comparison to their incoming rate..

Shortest Path: A->B->E->I

Shortest: 338

	Processing Delay	Transmission Delay	Buffer Size	Incoming Rate	Outgoing Rate
A	15	40	n/a	n/a	55
B	12	27	52	55	39
E	12	53	12	39	65
I	20	60	13	65	80

For the shortest path, loss can occur in nodes E and I.

Modified A* Path: A->B->F->D->H->K

Modified: 444

	Processing Delay	Transmission Delay	Buffer Size	Incoming Rate	Outgoing Rate
A	15	40	n/a	n/a	55
B	12	27	52	55	39
F	11	31	48	39	42
D	14	26	57	44	40
H	12	23	59	40	35
K	13	24	56	35	37

For the Modified A* path, loss can occur in nodes F and K.

Graphical Results

We can now graph the amount of loss that occurs for any given data packet, and the number of packets that are required to be sent through the path before loss occurs.

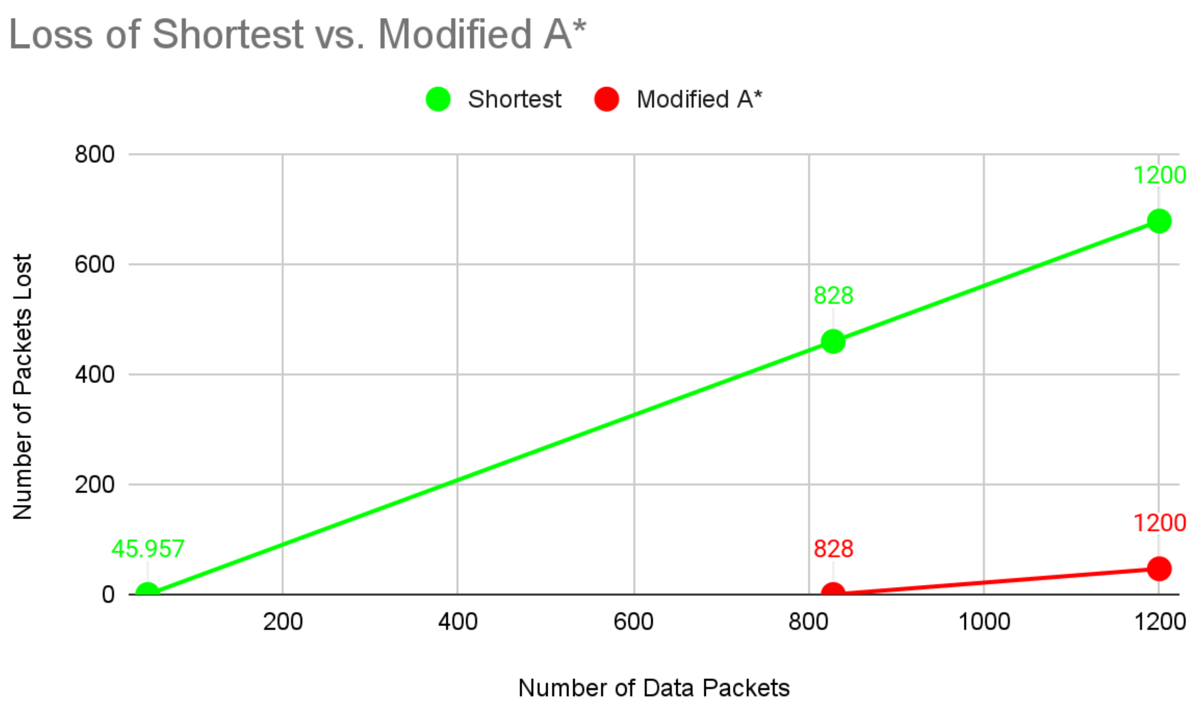


Figure 3.2: Graph of Loss for shortest path through default graph vs loss through modified A path*

Notice the points at which loss occurs, for the shortest path or a worst case scenario for packet loss, we begin losing packets very early on, when sending anything more than 45 packets through the network.On the other hand, routers optimized to avoid loss don’t begin to see packet loss until the number of packets being sent reaches 828 packets.

This is a significant difference between the two paths, as for any user attempting to send more than 45 packets through will experience loss, and the loss will increase at a greater rate than the modified A* path as well.

We can now apply this loss to the throughput calculations. The x-axis represents again the number of packets that we will be sending through the mesh network, with the y-axis representing the throughput. The throughput here is measured as before, with the x number of packets divided by the time taken for each packet, but we now graph the number of packets subtracting the number of packets lost. This throughput measurement is the number of packets that successfully reach the destination node.

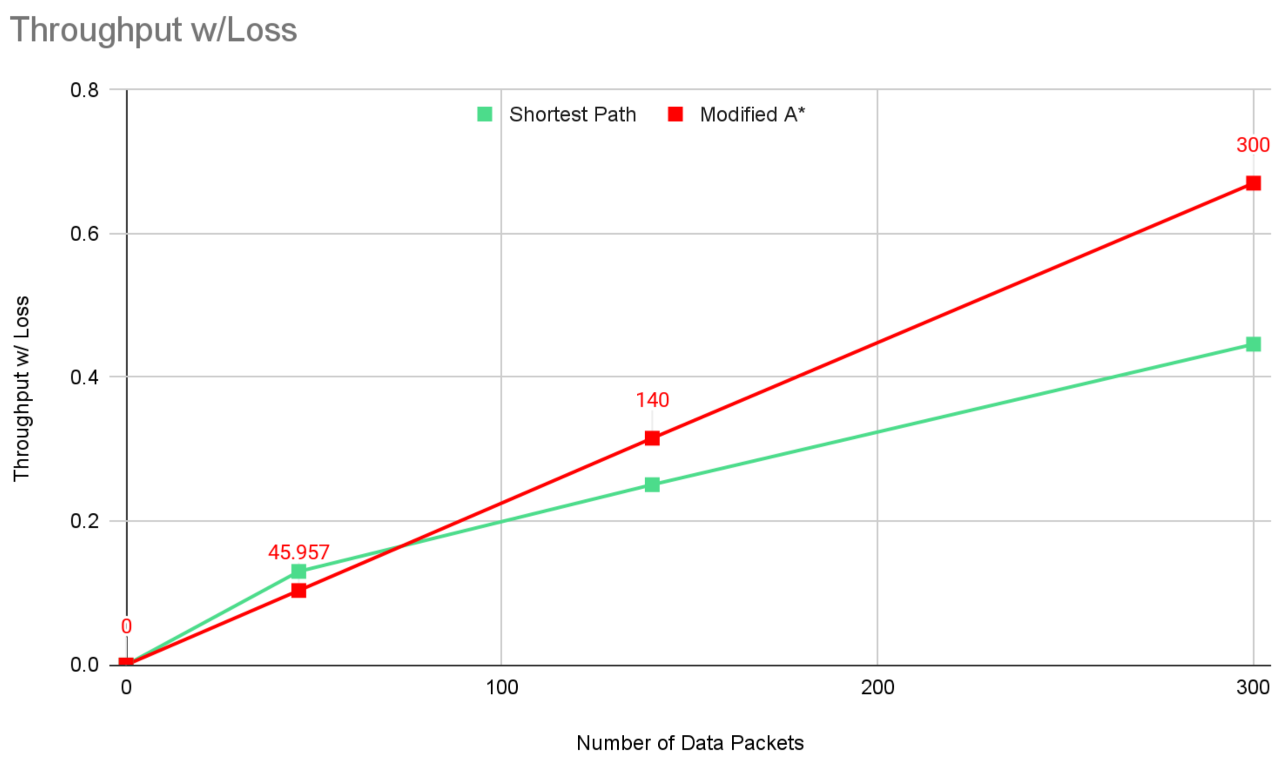


Figure 3.3: Graph of the throughput of the shortest path and the modified A path.*

On analysis, we can notice that at the start of the algorithm the shortest path has higher throughput, but at 45 packets, it will begin to experience loss, and it's throughput begins to decrease. In comparison, the A* path maintains a constant linear throughput even as the number of packets increases. At 77 packets, we see the two lines intersect, as the throughput of of the modified A* path grows to be larger than the shortest path due to the number of packets it is losing enroute to the final destination node.

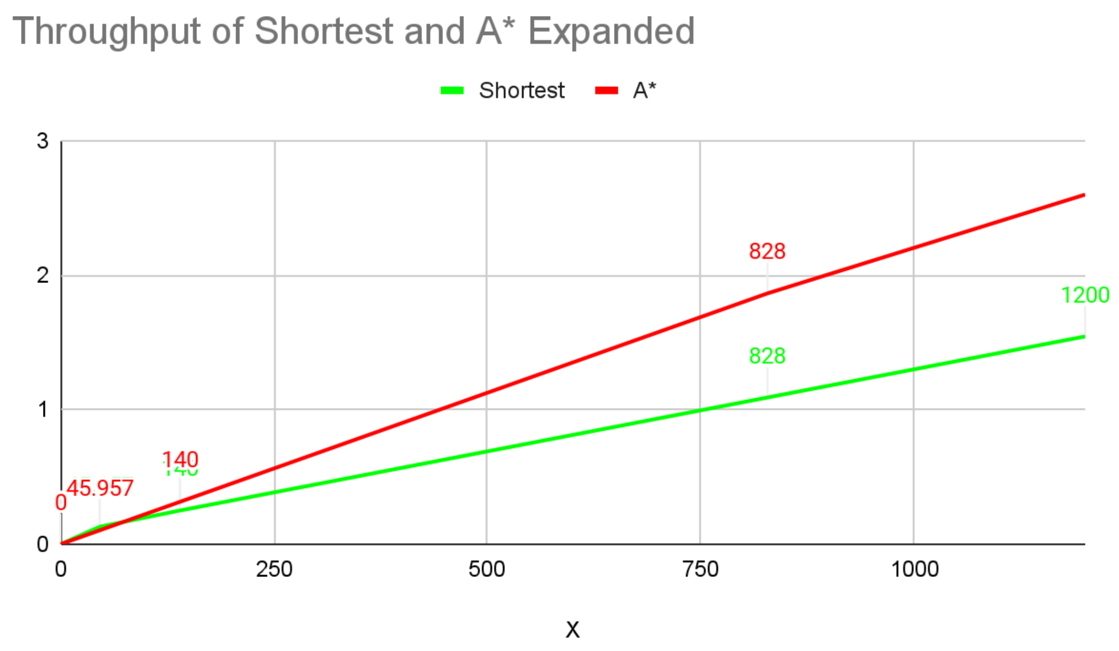


Figure 3.4: Expanded graph of shortest path and A throughput*

For an expanded view, notice that the throughput for A* only begins to decrease at 828, when loss is expected to occur. We can also tell that the linear slope for the A* is greater than the slope of the shortest path in both graphs of throughput. This corresponds to the slopes of the loss for each path. The rate at which the shortest path experiences loss is greater, so the rate at which it's throughput grows is less than A*'s, which experiences growth at a slower rate.

Conclusions

In conclusion, we can say that the path that our searching strategy found using the modified A* searching algorithm to be a generalized searching strategy. At a low number of packets, the path the algorithm will find will be worse than finding the shortest path, but in general for any given packet size, we can see that the A* strategy will find a path that has a higher throughput as the number of packets being sent through the network increases. By taking into account loss avoidance, we can optimize for throughput over a range of incoming packets.

Github Repository

---

https://github.com/jhowell702/CPE400-Fall-MeshNetwork.git