fybrrLink: Efficient QoS-aware routing in SDN enabled Space Architecture

Debajyoti Halder - Prashant Kumar - Saksham Bhushan

Abstract

- Demand for seamless anywhere anytime communication is increasing
- Satellite networks must be used as a service.
- Traditional IP-based architecture is unable to meet the current requirements.
- A variety of applications has diverse Quality of Service(QoS) requirements.
- Current satellite architecture doesn’t take these requirements into the account.
- We are proposing a novel and centralized QoS-aware routing algorithm for SDN enabled space network
- Further, fybrrLink is evaluated with multiple NS3 simulations...

Introduction

- Terrestrial network is not enough.
- Advantages of Software Defined Network(SDN)
- Using SDN for non-terrestrial architecture
- Variety of applications have diverse QoS requirements.

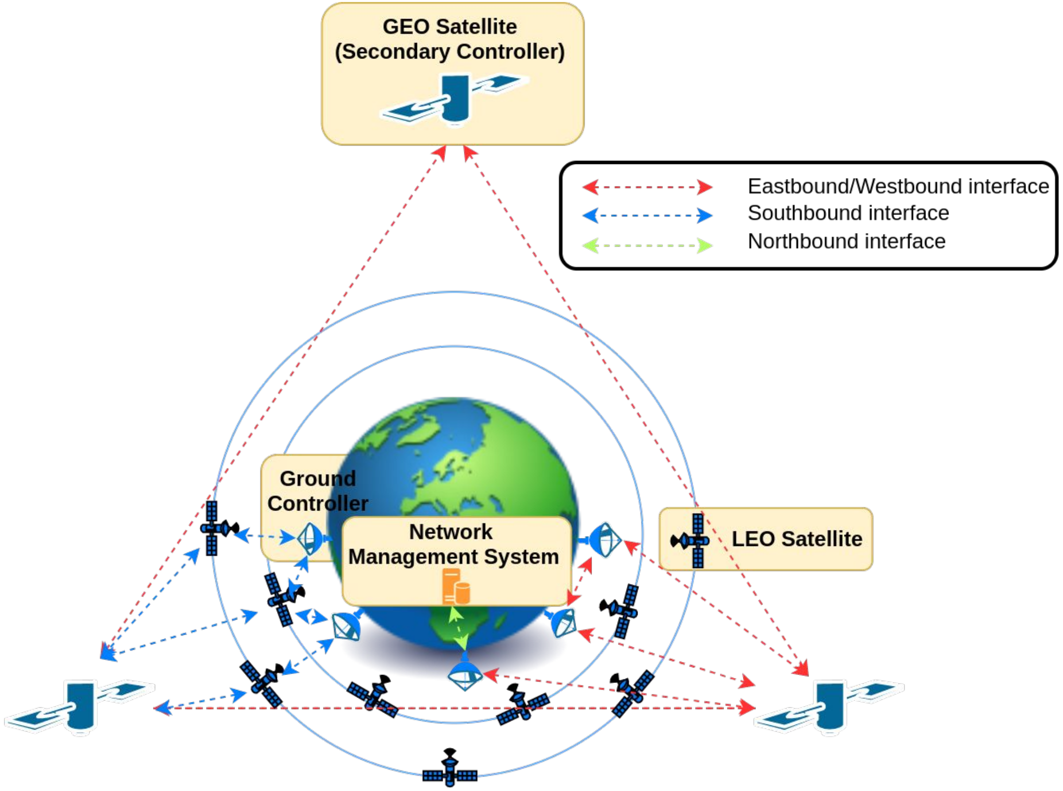
Contributions

- Extension of SDN to non-terrestrial network.
- Efficient QoS aware routing algorithm for space network-ing which computes better routes in much lesser time than the algorithms of other schemes.
- Avoidance of congested links during routing
- Quick rerouting during ISL congestion or satellite failure.
- Flow rule transfers for non-disruptive service during satellite handovers

Related Works

- Shortcoming of traditional satellite network
- Multiple SDN based architectures are being proposed
  - Software-Defined Next-Generation Satellite Networks:
  - STIN
  - SAGECELL
- Ant colony optimization
- Software-defined routing algorithm (SDRA) for NTN
- Software-defined space networking (SDSN)
- Well researched problem for terrestrial networks.....

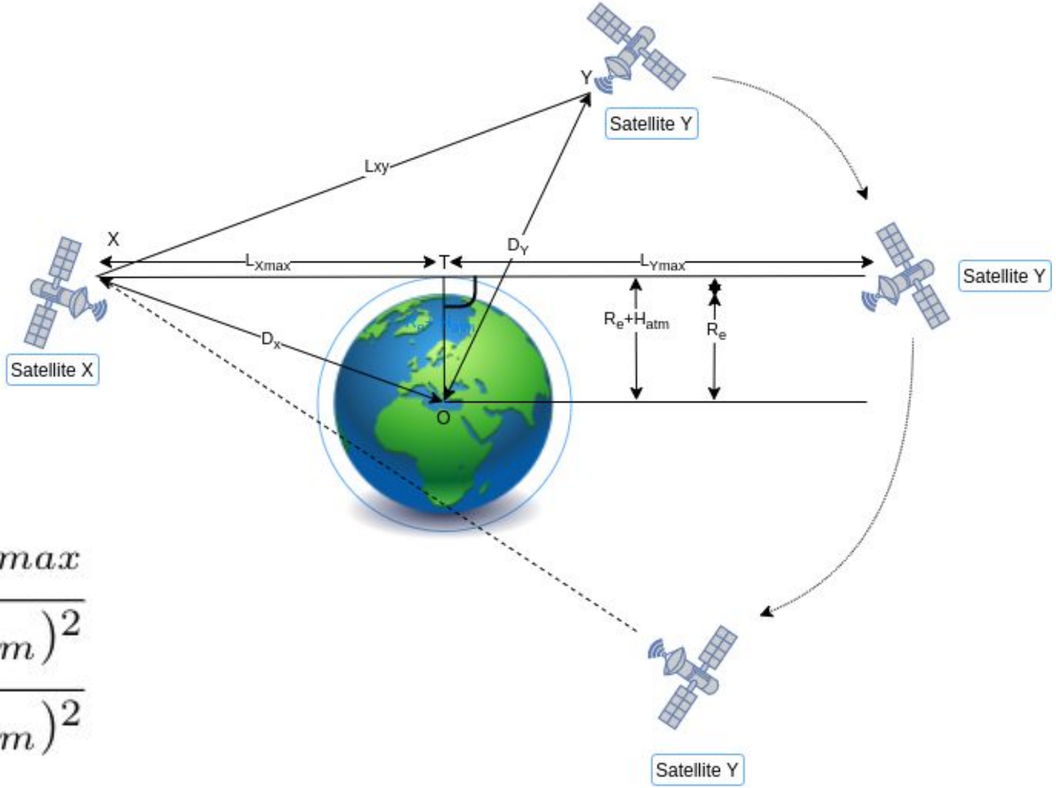
System Model



3 Modules

- Topology discovery and monitoring
  - To construct topology and monitor the link parameters
- Flow rule transfer
  - To Hande topology alteration problems
- QoS aware routing :
  - To calculate the most optimal path for a flow

Visibility of satellites



$$L_{XY} < L_{Xmax} + L_{Ymax}$$

$$L_{Xmax} = \sqrt{(D_X)^2 - (R_e + H_{atm})^2}$$

$$L_{Ymax} = \sqrt{(D_Y)^2 - (R_e + H_{atm})^2}$$

Communication Links and Cost Calculation

- Intra-Orbit Links
- Inter-Orbit Links
- A flow can have the following QoS parameters along with source and dest.
  - Minimum required bandwidth B
  - Maximum end to end delay D
  - Maximum jitter J
  - Maximum packet loss ratio PLR
- The fitness score of each edge can be defined as

  Score= k1*AB + k2/latency + k3*(1- PLR) + k4/Jitter + k5*StabilityFlag

- Further, Cost of the link between vertex i and vertex j is

  Cost[i,j] = 1/Score[i,j]

Link Parameters....

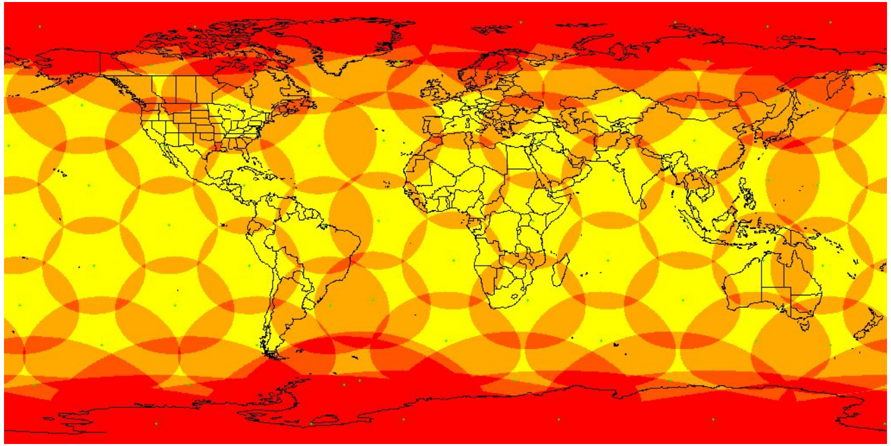
- B[i,j] represents the bandwidth (capacity) of link_{i,j}
- CD[i,j] is the congestion degree
    - CD[i,j] = Load[i,j]/B[i,j]
- D[i,j] is the delay associated with the link_{i,j}
- PLR[i,j] is the packet loss ratio of the link_{i,j}
    - PLR[i,j] = (PacketTransmitted[i,j]-PacketReceived[i,j])/PacketTransmitted[i,j]
- AB[i,j] represents the available bandwidth of the link
    - If Congested link if Load[i,j] is more than the 80% of B[i,j] or CD[i,j] > 0.8:
        - Put AB[i,j] = 0
    - Else:
        - AB[i,j] = B[i,j] - Load[i,j]
- J[i,j] is the average jitter associated with the link[i,j]
    - J[i,j] = 1/(t-startTime)*∑(|Latency[i,j][t(i)] - Latency[i][j][t(i+1)])... For startTime < i < t
- Stability Flag[i,j] = 1 if it is a intraorbit link and 0 otherwise

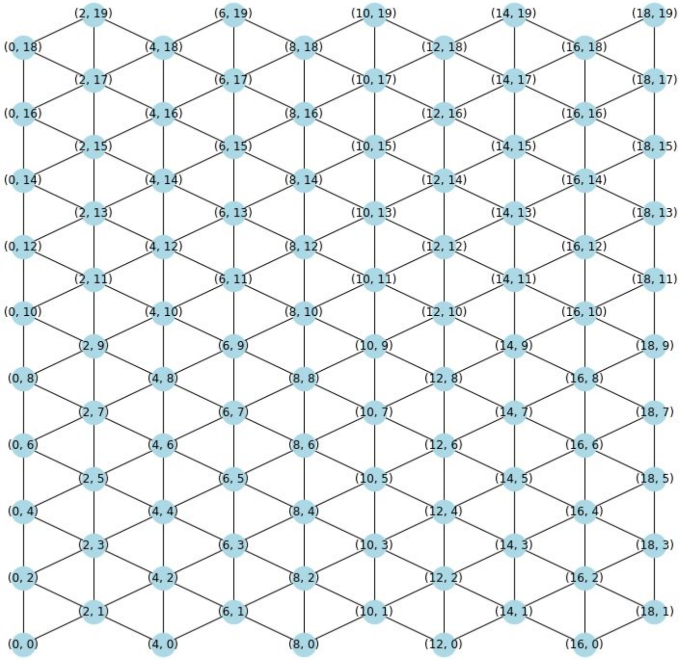
Proposed Approach

fybrrLink

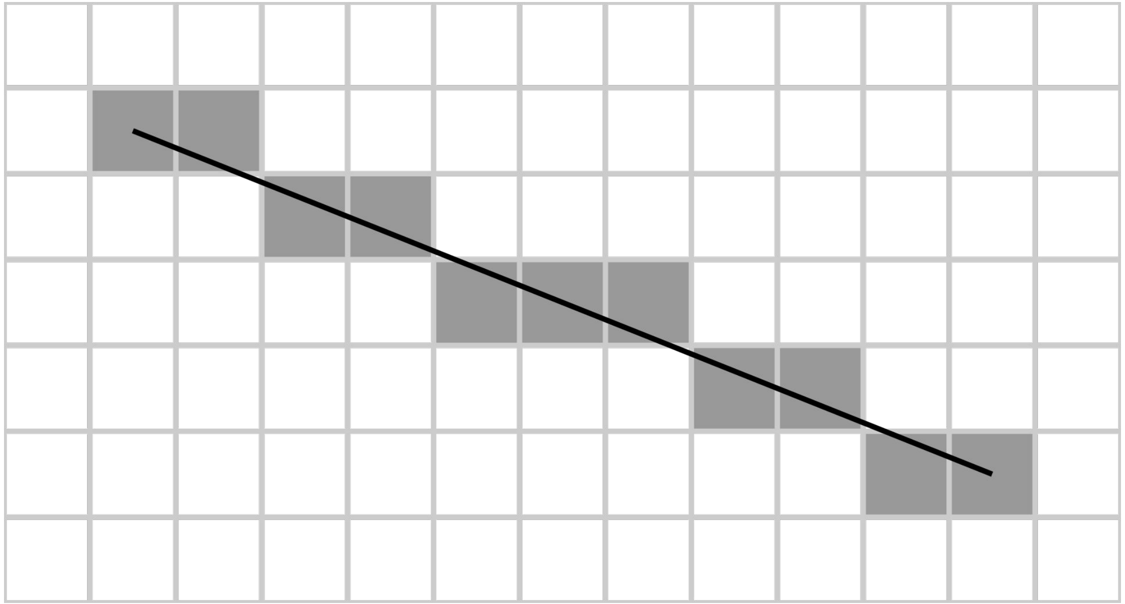
Assumptions

- Iridium-like satellite constellation - pole to pole orbit
- Satellites are connected in two different ISL types with total 6 ISLs
    - 2 Vertical ISLs
    - 4 Diagonal ISLs
- All satellite orbits are at same altitude from sea level



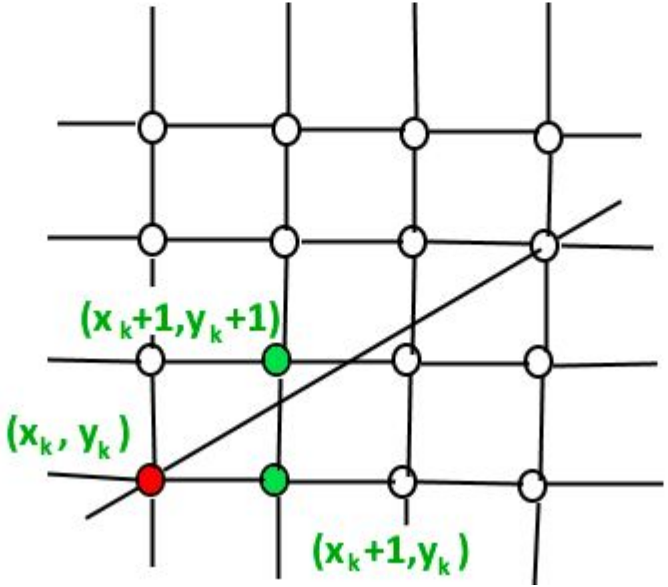


Bresenham's Line Algorithm

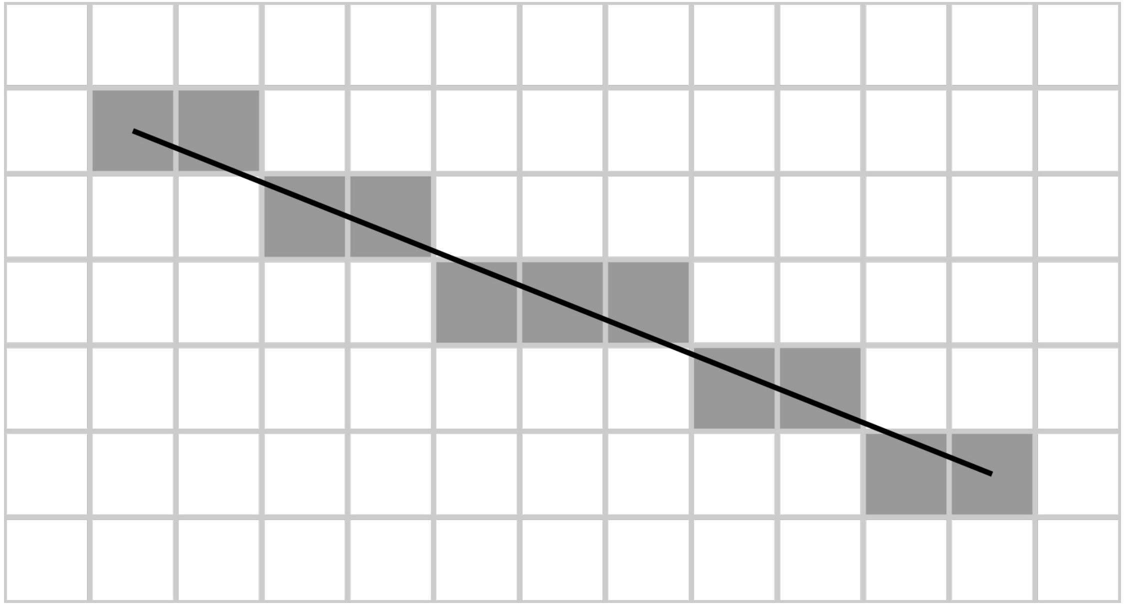


Determines points of an *n*-dimensional raster to be selected in order to form a close approximation to a straight line between two points.

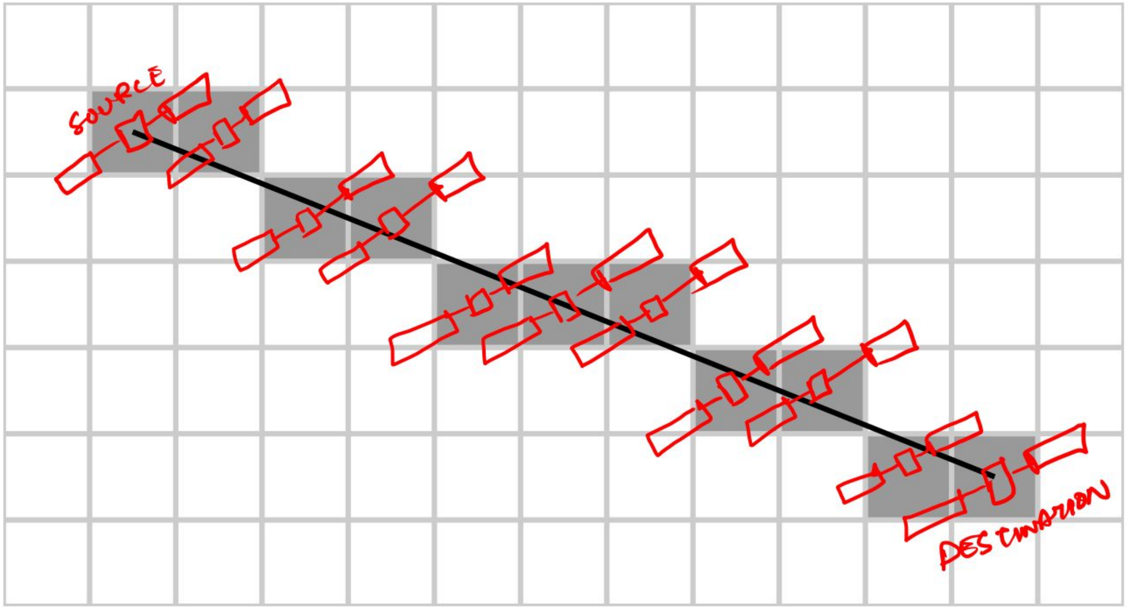
Routing using Bresenham's Line Algorithm



Routing using Bresenham’s Line Algorithm

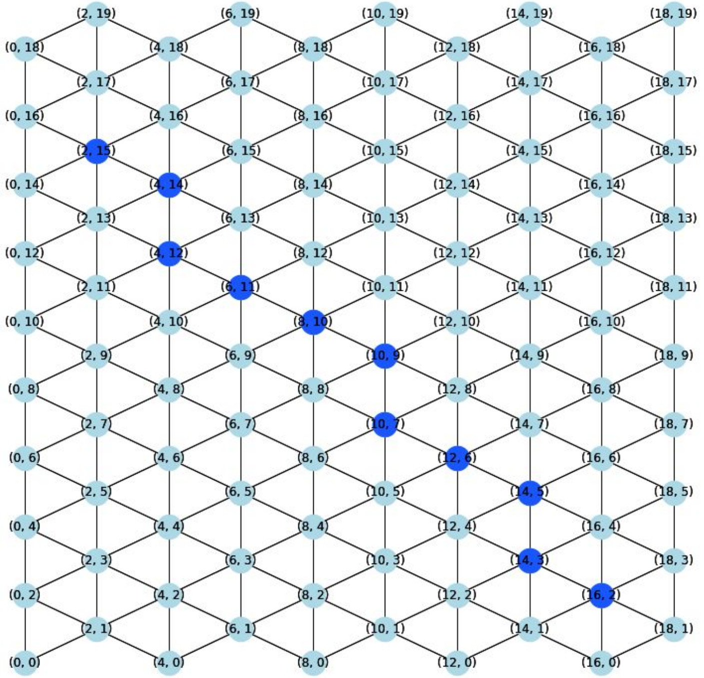


Routing using Bresenham's Line Algorithm



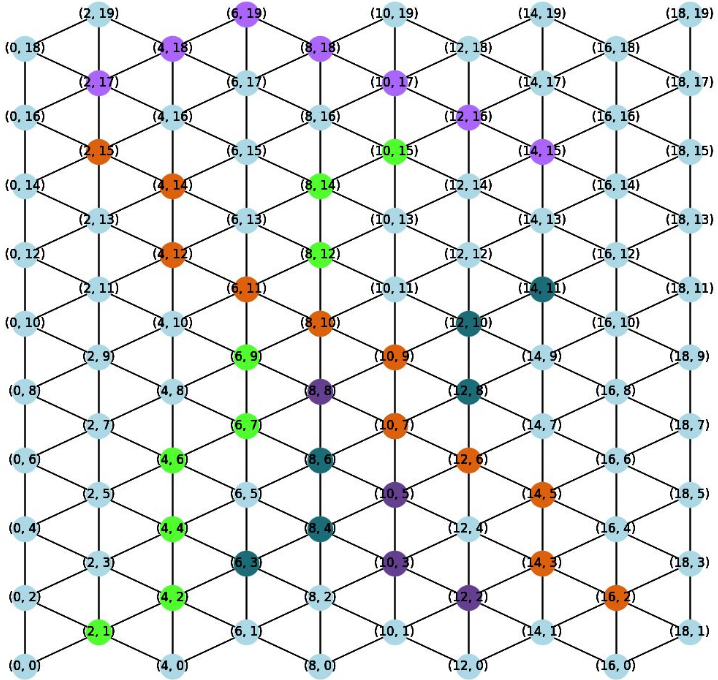
Modified Bresenham’s Routing Algorithm

```
def routingFunction(G, F):
    for f in F:
        source = G.XYCoordinate(f.source)
        dest = G.XYCoordinate(f.dest)
        m, n, initial_shortest_path = bresenham(G, source, dest)
        ''' m = horizontal steps,
            n = vertical steps and
            initial_shortest_path = shortest path produced by bresenham algorithm
        '''
        if f.QoSenabled == False or isSatisfyingPath(initial_shortest_path, f.flow_constraints):
            updateFlowTable(f, initial_shortest_path)
        else:
            search_graph = getSmallerSearchGraph(G, source, dest, m, n)

            # Congestion avoidance
            maxCD = 0.85 # maximumAllowedCongestionDegree
            deltaCD = 0.05
            no_of_iter = (0.96 - maxCD)/deltaCD

            isSuitablePathFound = False
            for i in range(no_of_iter): #
                path = DijkstraAlgo(search_graph, source, dest)
                if isSatisfyingPath(path, f.flow_constraints):
                    updateFlowTable(f, path)
                    isSuitablePathFound = True
                    break
                else:
                    CD += deltaCD # deltaCD
                    assignCost(Graph, CD, source)

            if not isSuitablePathFound:
                informSender("Message":"Given Flow cannot be satisfied. Please relax some contraints",
"BestPath":path)
```



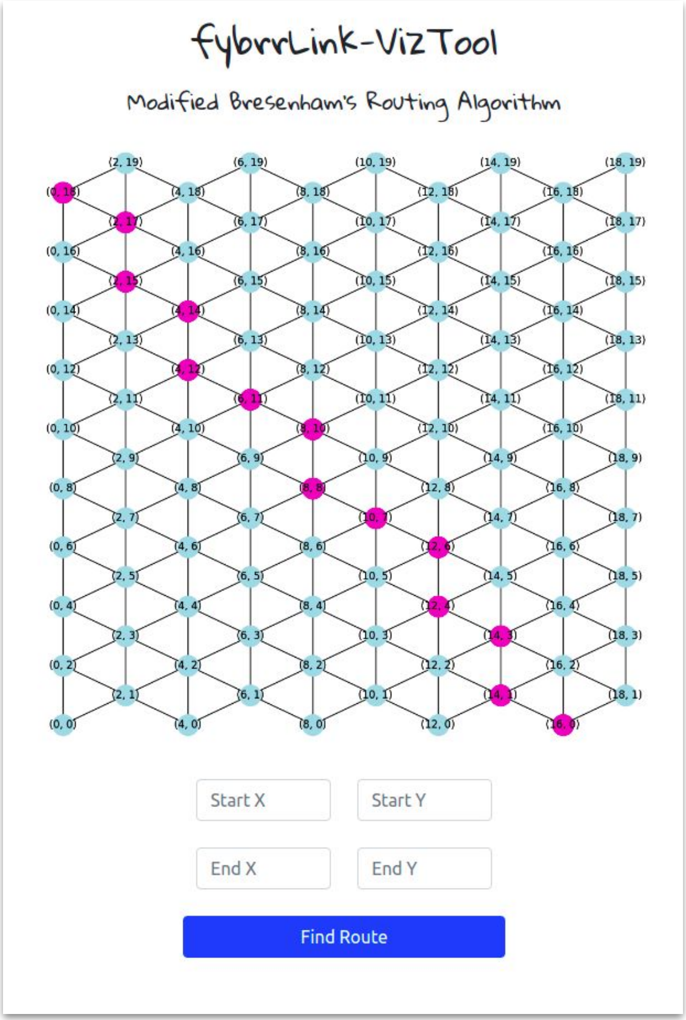
NetworkX - Used for visualization of the algorithms

```python
def graph_init():
    size = 20
    sz = int(size / 2)
    G = nx.grid_2d_graph(sz, sz)
    pos = list(G.nodes())
    for i in range(sz * sz):
        if pos[i][0] % 2 != 0:
            pos[i] = (pos[i][0], pos[i][1] + 0.5)
        pos[i] = (pos[i][0] * 2, int(pos[i][1] * 2))

    G_new = nx.Graph()
    plt.figure(figsize=(sz, sz))
    pos1 = {(x, y): (x, y) for x, y in G_new.nodes()}

    link = ISL(1)

    G_new.add_weighted_edges_from([…

    nx.draw(G_new, pos=pos1,
            node_color='lightblue',
            with_labels=True,
            node_size=600)
    nx.draw(G_new, pos=pos1, node_color="lightblue", with_labels=True, node_size=600)
    for i in node_rem:
        color = "%06x" % random.randint(0, 0xFFFFFF)
        color = "#"+color
        nx.draw(G_new, pos=pos1, nodelist=i, node_color=color, with_labels=True, node_size=600)
        labels = nx.get_edge_attributes(G_new,'weight')
        nx.draw_networkx_edge_labels(G_new,pos1,edge_labels=labels)

    plt.show()
```

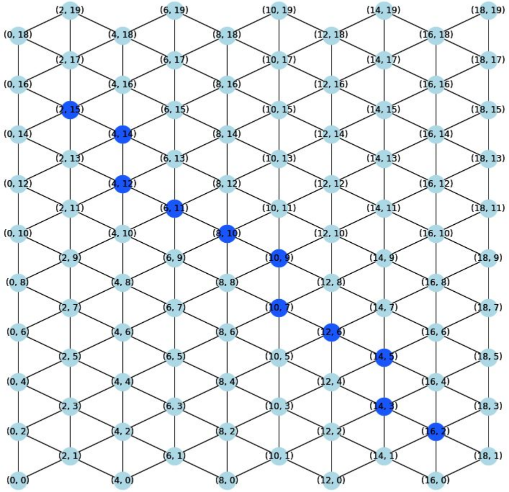


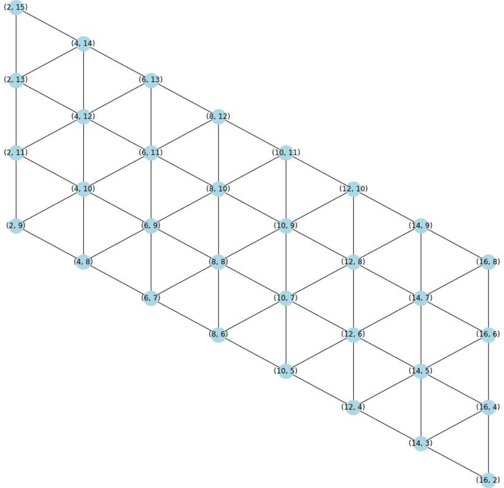
fybrrLink-VizTool

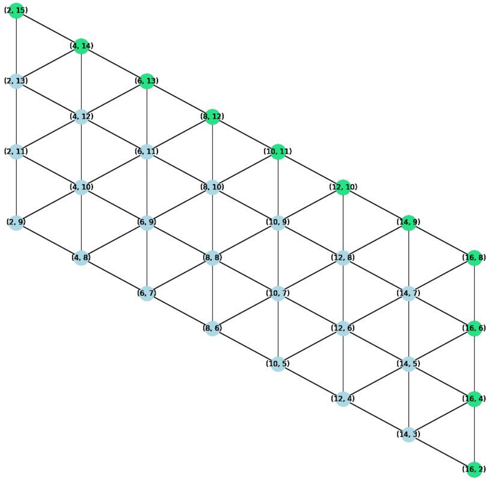
- Implementation of the modified bresenham algorithm is available as a web tool.
- Link : https://fybrrlink-viztool.herokuapp.com



Flow of the fybrrLink Algorithm



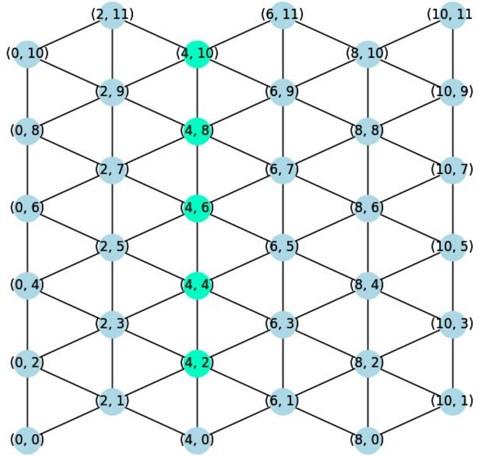




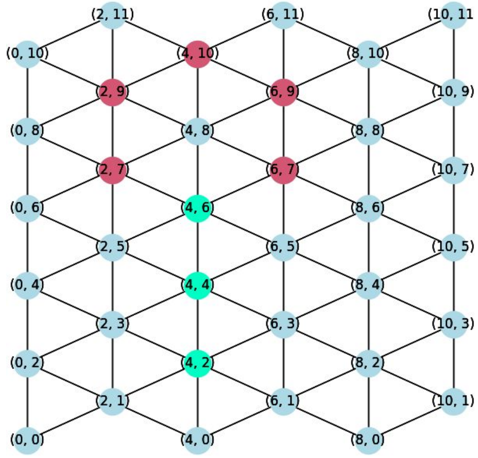
Edge Cases

Direct Vertical Path

- The shortest path between source and destination satellites comprises vertical ISLs only.
- Cannot form Ideal Parallelogram
- Deviate path during congestion and run routing algorithm to find route from possible intermediate satellites.



No congestion

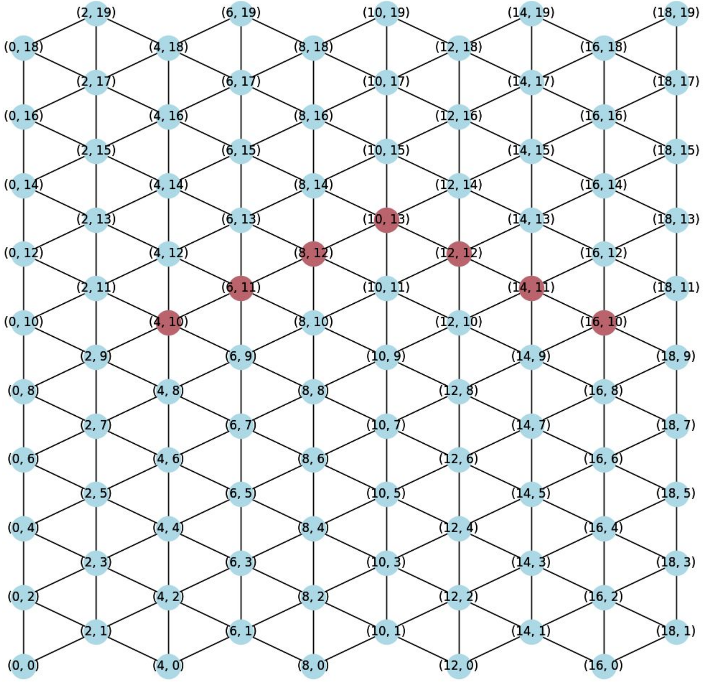


After congestion

Edge Cases

Only Diagonal ISLs Path

- The shortest path between source and destination satellites comprises diagonal ISLs only.
- Ideal Parallelogram can be formed with mirrored links

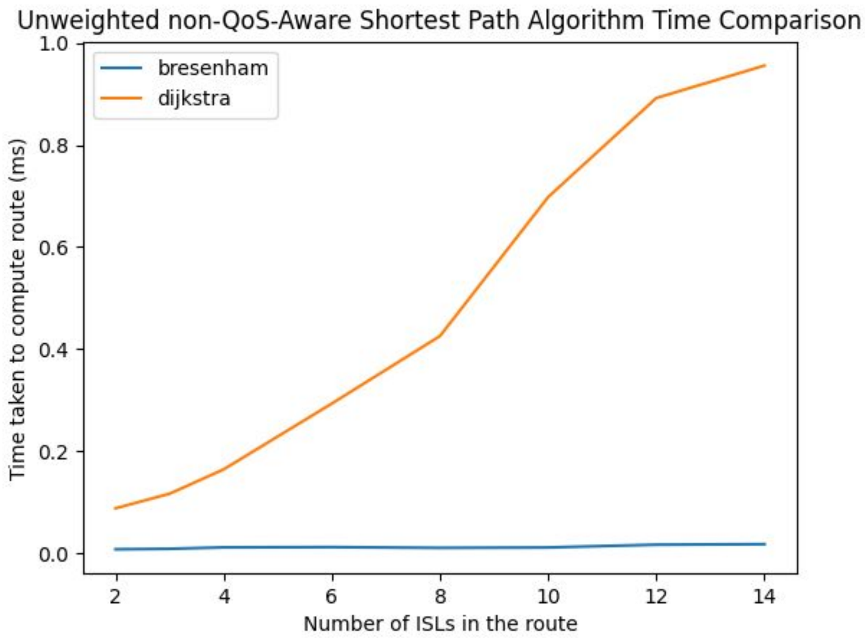


Complexity Analysis

Time Complexity

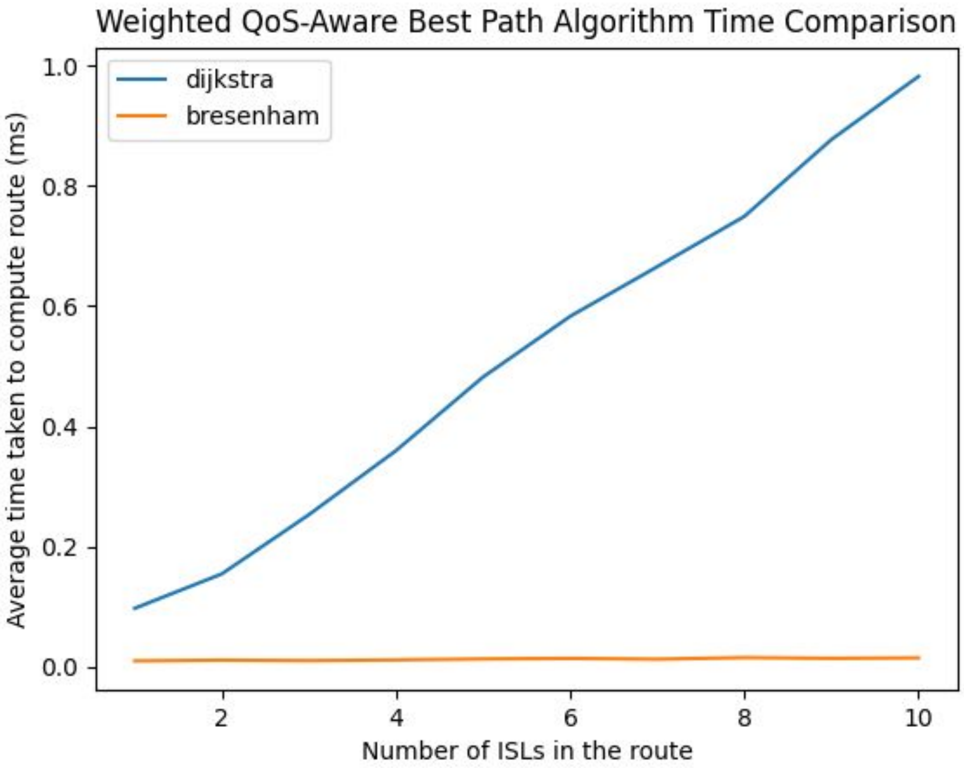
- Dijkstra - O(V + ElogV)
- Bresenham - O(N)
    - N = m + n
    - m ➔ vertical ISLs
    - n ➔ diagonal ISLs
- fybrrLink - O(N) + O(V' + E'logV')
    - V' ➔ (m + 1)(n + 1)
    - E' ➔ (m + 1)n + (n + 1)m

Complexity Analysis



	ISLs	bresenham	dijkstra
0	2	0.007629	0.087976
1	3	0.008583	0.116587
2	4	0.011444	0.164270
3	6	0.011921	0.293255
4	8	0.010490	0.425577
5	10	0.011206	0.697851
6	12	0.016689	0.891924
7	14	0.017643	0.955582

Complexity Analysis



Performance Evaluation

What we have implemented?

1. Modified bresenham algorithm
2. Conversion of satellite coordinates to 2d coordinates
3. Graph implementation of satellite network
4. Implementation of the function getIdealParallelogram()
5. Implementation of the function getEdgeCost()

Performance Metrics

1. Average execution time
2. Packet Loss rate
3. Average Latency/End-to-end Delay
4. Congestion degree
5. Satisfaction Ratio

Experimental Setup

Flow control Simulation - NS3 Network Simulator.

Routing Algorithm Simulation - Networkx on Python.

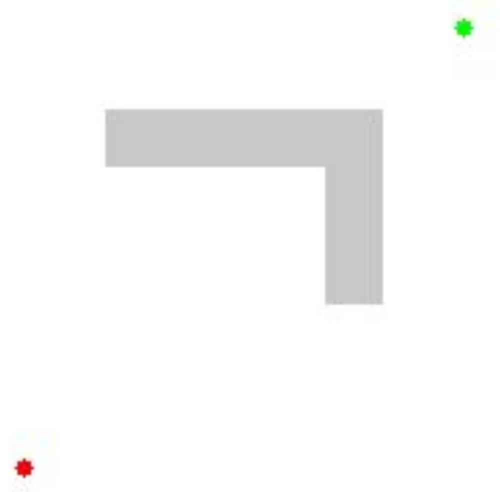
SIMULATION PARAMETERS

Parameter	Values
Number of test cases (flows)	324
Simulation Time	324 seconds
Incoming flow frequency	1 flow per second
Simulator	NS 3.30.1
Number of satellites per orbit	10
Number of orbits	10
Orbit Altitude	2000 Km
maxCD (for Algo. 1)	0.82
deltaCD (for Algo. 1)	0.05
Allotted bandwidth for Intra-Orbit link	4.16 Gbps
Allotted bandwidth for Inter-Orbit link	(3.10 - 3.70) Gbps
Initial Congestion degree	0.60 - 0.80
Initial Packet loss rate	0.0001 - 0.002
Latency of Intra-Orbit links (at t = 0)	17.5592 ms
Latency of Inter-Orbit links (at t = 0)	(2.7029 - 19.3630) ms
$[k_1, k_2, k_3, k_4, k_5]$	[0.55, 0.30, 0.15, 0, 0]

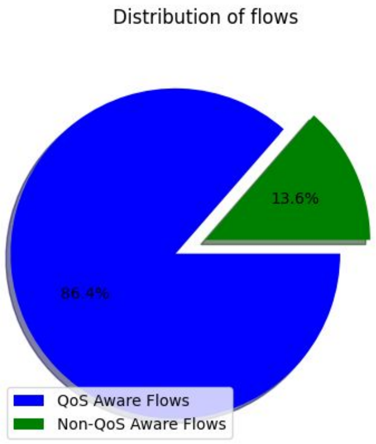
Schemes to compare

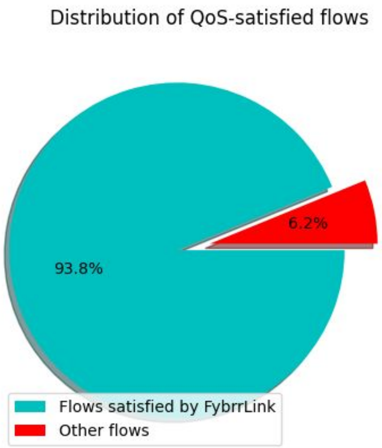
Dijkstra Algorithm

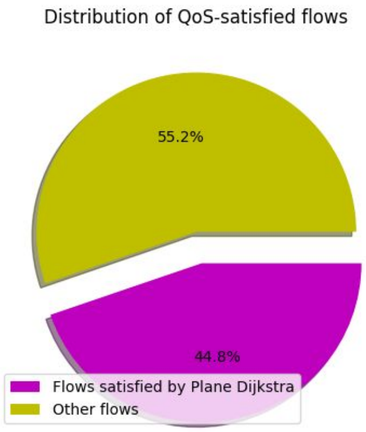
- The most popular shortest path finding algorithm.
- Compared with almost every NTN based routing proposals.



QoS-aware flow distributions

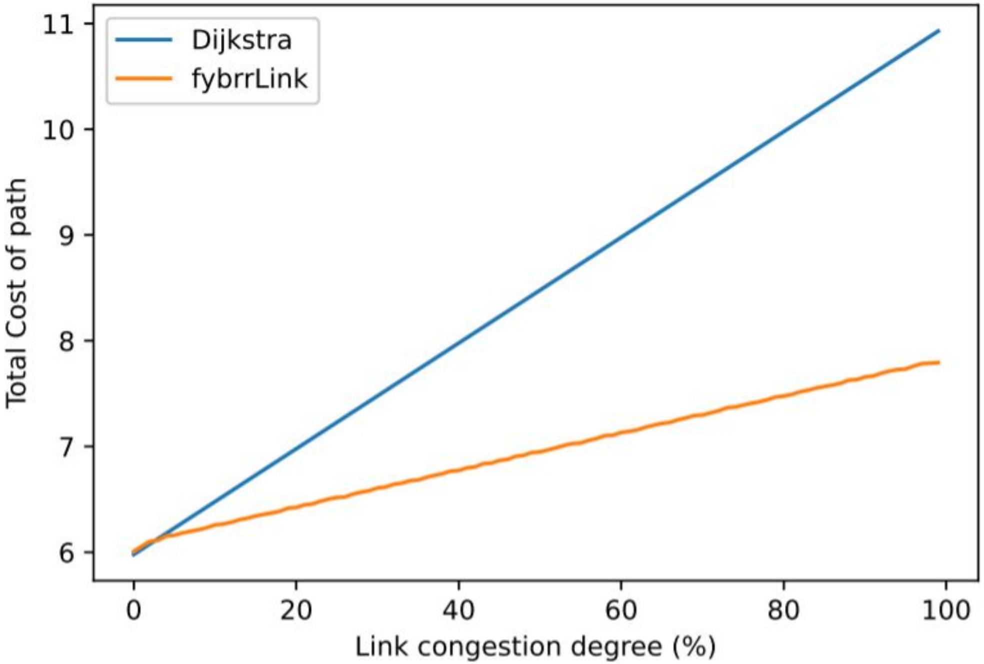






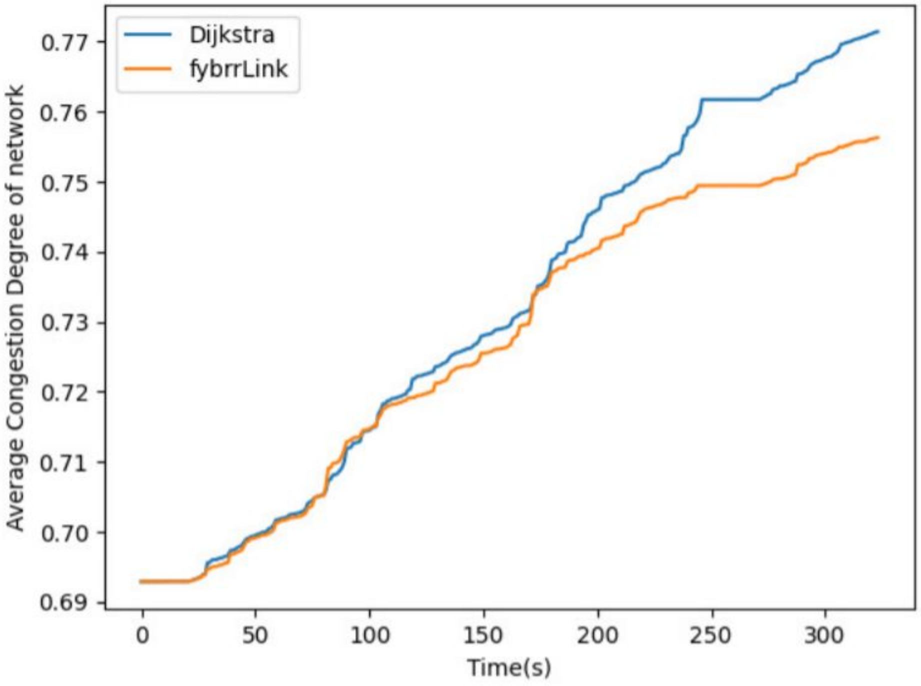
Congestion Degree analysis

- Cost of path with increasing link congestion degree.
- fybrrLink performs 27% better than Dijkstra.



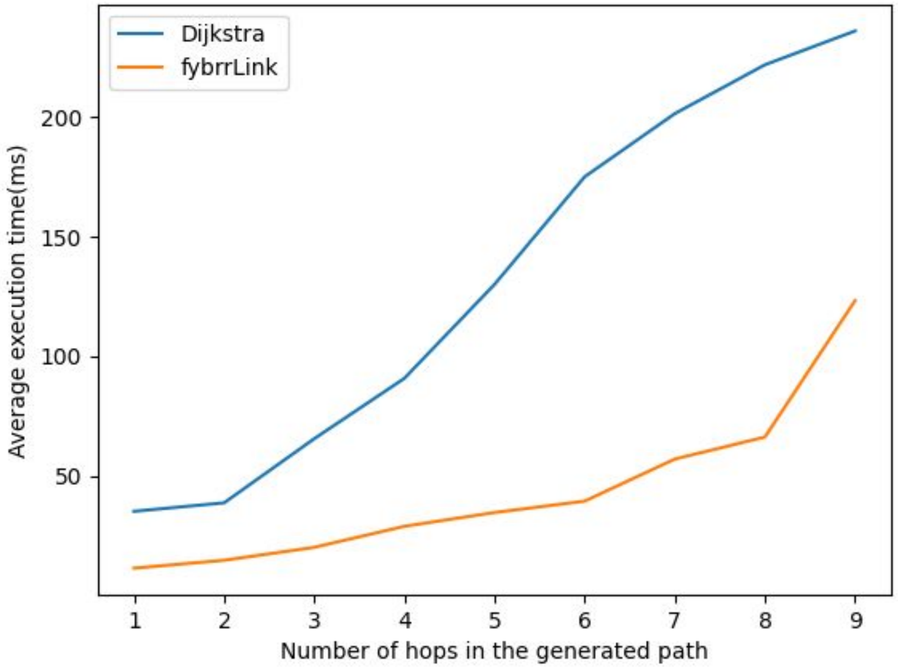
Variation of Average CD with time

- Congestion degree trend with incoming packets with rate 1 per sec.
- fybrrLink performs better than Dijkstra.



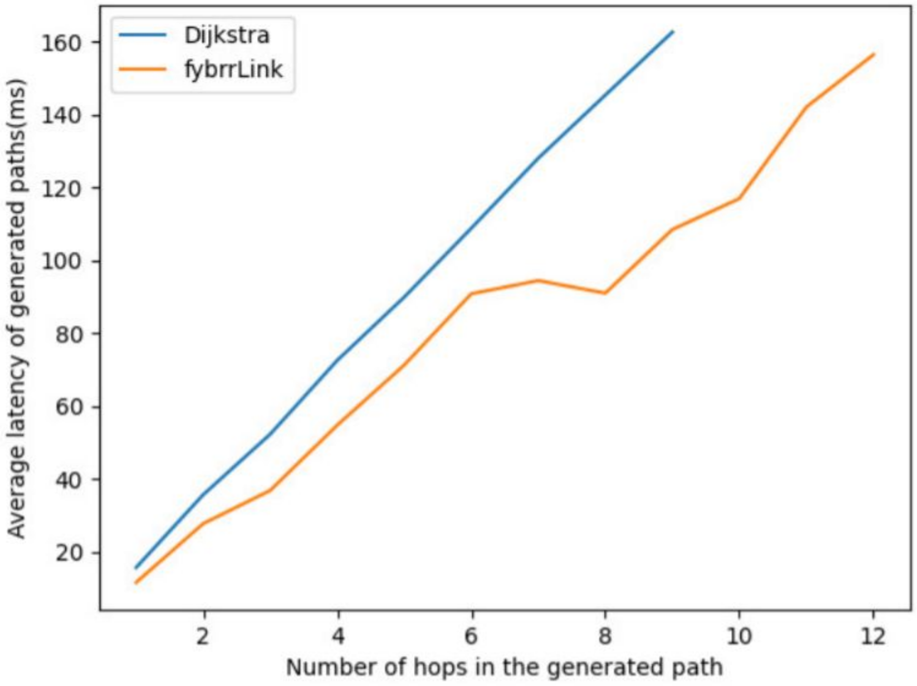
Average Execution time vs #hops in path

- Time taken for computing path with increasing search space.



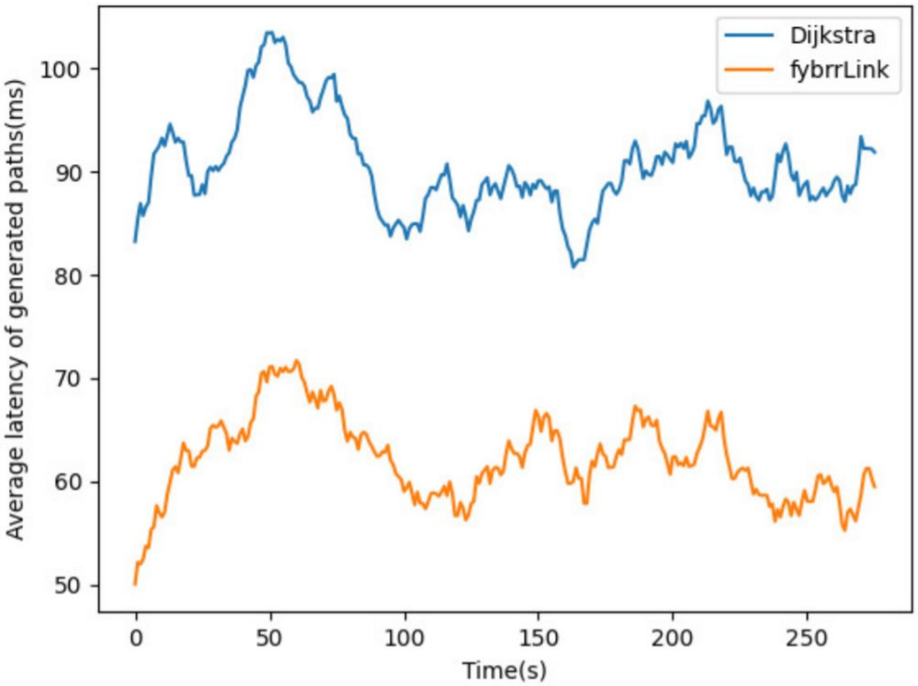
Average latency of paths vs #hops in path

- fybrrLink finds path with lower latency than Dijkstra.
- For 8 hops, fybrrLink performs 38% better than Dijkstra.



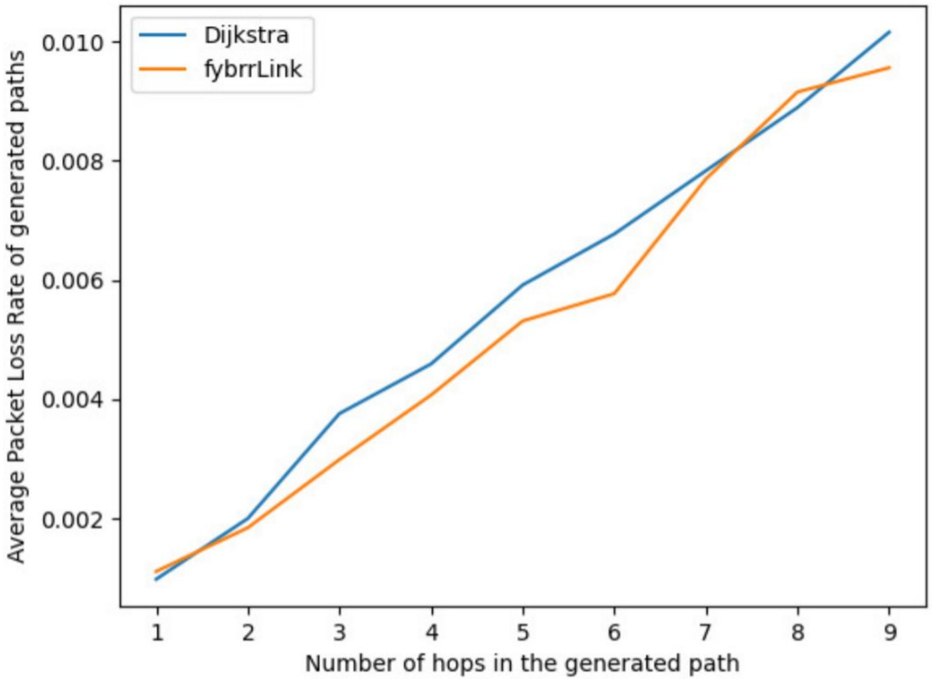
Variation of average latency of paths (moving average) with time.

- Moving average of window 50 for simulation duration.
- Clearly, fybrrLink is able to find paths with lower latency for all test flows.



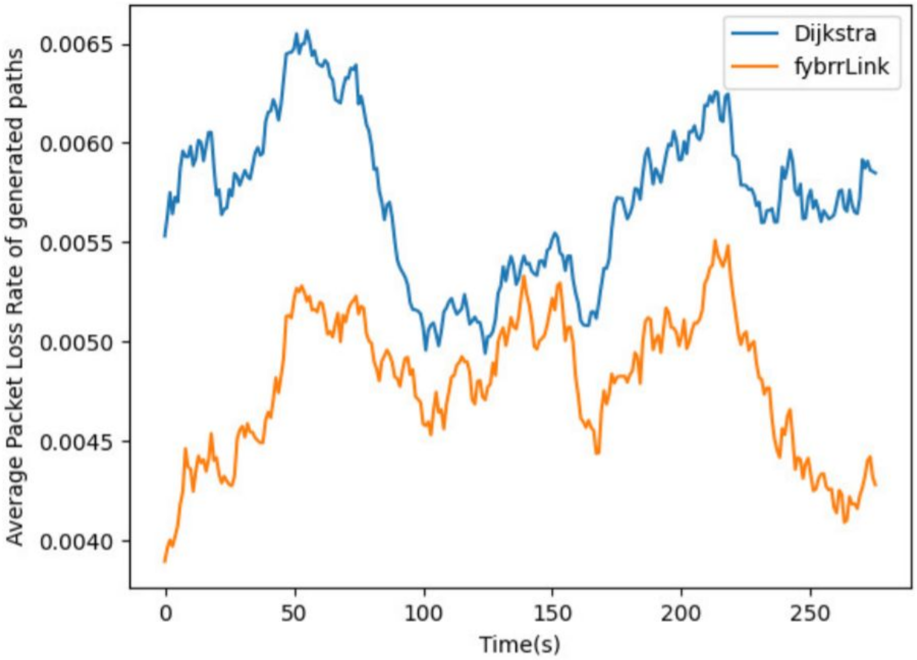
Average PLR of paths vs #hops in path

- Increasing PLR with increasing hops.
- fybrrLink performs slightly better than Dijkstra.



Variation of average PLR of paths (moving average) with time.

- Moving average of window 50 for simulation duration.
- Clearly, fybrrLink is able to find paths with slight lower PLR for all test flows.



Conclusion

- Proposed QoS-aware routing algorithm using the global knowledge from SDN in Non-Terrestrial Networks.
- Proposed 6 ISL link model for more efficient routing.
- Simulations on NS3 provides credibility to the comparison.
- Robust evaluation and comparison of our approach with Dijkstra proves fybrrLink to be better.

Future Works

- Integration of fybrrLink with the existing terrestrial network can be explored.
- More analysis can be done on Controller Placement Problem, terrestrial controllers can be compared with Non-terrestrial controllers.
- Including security aspects of the SDN and Non-Terrestrial Networks for QoS-aware routing can be an interesting research area.

References

[1] M. Handley, “Delay is not an option: Low latency routing in space,” in 17th ACM Workshop, 11 2018, pp. 85–91.

[2] D. Kreutz, F. M. V. Ramos, P. E. Ver´ıssimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, “Software-defined networking: A comprehensive survey,” Proceedings of the IEEE, vol. 103, no. 1, pp. 14–76, 2015.

[3] S. Xu, X. Wang, and M. Huang, “Software-defined next-generation satellite networks: Architecture, challenges, and solutions,” IEEE Access, vol. 6, pp. 4027–4041, 2018.

[4] L. Yang, J. Liu, C. Pan, Q. Zou, and D. Wei, “Mqrp: Qos attribute decision optimization for satellite network routing,” in 2018 IEEE International Conference on Networking, Architecture and Storage (NAS), 2018, pp. 1–8.

[5] Q. Guo, R. Gu, T. Dong, J. Yin, Z. Liu, L. Bai, and Y. Ji, “Sdn-based end-to-end fragment-aware routing for elastic data flows in leo satelliteterrestrial network,” IEEE Access, vol. 7, pp. 396–410, 2019.

[6] “Cisco collaboration system 12.x solution reference network designs (srnd),” 03 2018, [Online].

[7] Y. Bi, G. Han, S. Xu, X. Wang, C. Lin, Z. Yu, and P. Sun, “Software defined space-terrestrial integrated networks: Architecture, challenges, and solutions,” IEEE Network, vol. 33, no. 1, pp. 22–28, 2019.

[8] Z. Zhou, J. Feng, C. Zhang, Z. Chang, Y. Zhang, and K. M. S. Huq, “Sagecell: Software-defined space-air-ground integrated moving cells,” IEEE Communications Magazine, vol. 56, no. 8, pp. 92–99, 2018.

[9] D. Yan, J. Guo, L. Wang, and P. Zhan, “Sadr: Network status adaptive qos dynamic routing for satellite networks,” in 2016 IEEE 13th International Conference on Signal Processing (ICSP), 2016, pp. 1186–1190.

[10] L. Zhang, F. Yan, Y. Zhang, T. Wu, Y. Zhu, W. Xia, and L. Shen, “A routing algorithm based on link state information for leo satellite networks,” in 2020 IEEE Globecom Workshops (GC Wkshps), 2020, pp. 1–6.

[11] T. Pan, T. Huang, X. Li, Y. Chen, W. Xue, and Y. Liu, “Opspf: Orbit prediction shortest path first routing for resilient leo satellite networks,” in ICC 2019 - 2019 IEEE International Conference on Communications (ICC), 2019, pp. 1–6.

[12] Y. Zhu, L. Qian, L. Ding, F. Yang, C. Zhi, and T. Song, “Software defined routing algorithm in leo satellite networks,” in 2017 International Conference on Electrical Engineering and Informatics (ICELTICs), 2017, pp. 257–262.

[13] T. Xie, “Sdsn: Software-defined space networking — architecture and routing algorithm,” 10 2019. [14] W. Cho and J. P. Choi, “Cross layer optimization of wireless control links in the software-defined leo satellite network,” IEEE Access, vol. 7, pp. 113 534–113 547, 2019.

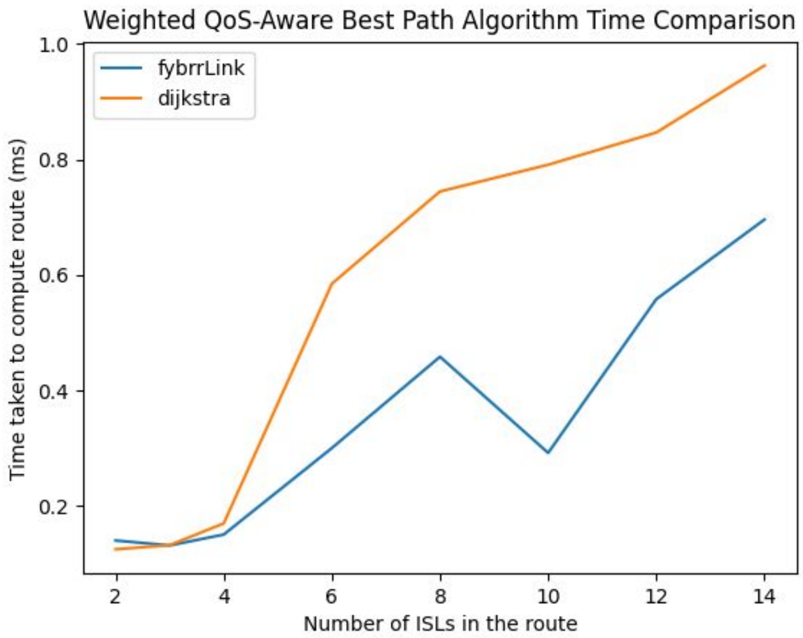
[14] W. Cho and J. P. Choi, “Cross layer optimization of wireless control links in the software-defined leo satellite network,” IEEE Access, vol. 7, pp. 113 534–113 547, 2019.

Thank you

ISL cost for QoS

```
class ISL:
    def __init__(self, link_type):
        self.link_type = link_type
        if self.link_type == 0:
            self.throughput = 100 - np.random.poisson(5)
            self.delay = 50 + np.random.poisson(5)
            self.congestion = 0
        elif self.link_type == 1:
            self.throughput = 100 - np.random.poisson(10)
            self.delay = 70 + np.random.poisson(7)
            self.congestion = 0
    def getScore(self):
        self.score = self.throughput/100 + 50/self.delay - self.congestion/100
        return self.score

    def getCost(self):
        self.cost = 2/self.getScore()
        return self.cost

    def setCost(self, weight):
        self.cost = weight

    def updateCongestion(self, value, weight = self.getCost()):
        self.congestion += value
```

Complexity Analysis



	ISLs	fybrrLink	dijkstra
0	2	0.140429	0.125170
1	3	0.131845	0.132322
2	4	0.150681	0.169992
3	6	0.300407	0.584841
4	8	0.458479	0.744581
5	10	0.292063	0.790834
6	12	0.558138	0.846624
7	14	0.695944	0.962496

Modified bresenham algorithm

1. where every integer point is having 8 degrees of freedom
2. but we developed a modified bresenham algorithm which will work fine with 6 degrees of freedom (2 intra-plane links and 4 inter-plane diagonal links) too.
3. We derived the bresenham once again with our own requirements to get this modified version

Conversion of satellite coordinates to 2d coordinates

1. Position of a satellite generally refers to its longitude, latitude, elevation angle, height etc.
2. but we mapped this to simpler x,y coordinates (assuming each satellite is revolving at same height only) for routing purposes.

Graph implementation of satellite network:

1. As per our system model, satellites were considered as nodes and their links as edges.
2. We implemented this class and required methods (e.g., getEdgeCost) so that routing algorithms can be performed on this graph topology.
3. getEdgeCost() : This function was required to assign cost to each link by taking its latency, bandwidth, packet loss etc. factors into the account.

Implementation of the function getIdealParallelogram():

1. In our routing algorithm, we were reducing the search space of the Dijkstra algorithm by finding a smaller size parallelogram.
2. Formation of that parallelogram by computing maximum possible diagonal and vertical links and
3. removal of unnecessary nodes/links from the graph was the goal for this particular function.