# Multiple DCLC Routing Algorithms for Ultra-Reliable and Time-Sensitive Applications

Piyush Navade, Lisa Maile, Reinhard German
Computer Networks and Communication Systems
Friedrich-Alexander-Universität
Erlangen-Nürnberg, Germany
{piyush.navade, lisa.maile, reinhard.german}@fau.de

***Abstract*—This paper discusses the problem of finding multiple shortest disjoint paths in modern communication networks, which is essential for ultra-reliable and time-sensitive applications. Dijkstra's algorithm has been a popular solution for the shortest path problem, but repetitive use of it to find multiple paths is not scalable. The Multiple Disjoint Path Algorithm (MDPAlg), published in 2021, proposes the use of a single full graph to construct multiple disjoint paths. This paper proposes modifications to the algorithm to include a delay constraint, which is important in time-sensitive applications. Different delay-constraint least-cost routing algorithms are compared in a comprehensive manner to evaluate the benefits of the adapted MDPAlg algorithm. Fault tolerance, and thereby reliability, is ensured by generating multiple link-disjoint paths from source to destination.**

***Index Terms*—Dijkstra's algorithm, shortest path routing, disjoint multi-paths, delay constrained, least cost.**

## I. Introduction

With the increasing demand for ultra-reliable and time-sensitive applications in modern communication networks, it has become necessary to develop routing algorithms that can ensure reliable and fast delivery of data packets. Traditionally, the shortest path problem has been a well-known problem in graph theory, Dijkstra's algorithm [1], being the most popular solution. To ensure ultra-high reliability, this problem is extended to finding multiple paths that do not share any common links, so that in the event of a failure, the transmission of data can still be guaranteed. Many solutions which use Dijkstra's algorithm for this problem have been proposed in the literature, like [2] and [3]. However, they require the repetitive use of Dijkstra's algorithm to generate multiple paths, which is not a scalable solution [4].

Recently, Multiple Disjoint Path Algorithm (MDPAlg) [5] proposes a novel method involving a single full graph instead of using Dijkstra's to construct multiple disjoint paths. When time-sensitive applications are considered, the generated disjoint paths should also keep the delay constraint, which is why we propose modifications to the algorithm of [5]. First, we start with modifications to simple Dijkstra's algorithm to make it scalable when the number of nodes increases in a network and then similar modifications are made to MDPAlg, to include the delay constraint.

It is to be noted that the proposed solutions in this paper are solutions to a routing problem and not an optimization problem. Generally, optimization problems in computer networks involve maximizing or minimizing a certain metric, such as bandwidth utilization, while considering various constraints. Here, we are only concerned with determining the most efficient path for data packets to travel from one node to another while considering the cost of the path and its delay.

In summary, the contributions of this paper are as follows: (i). Evaluating the efficiency and scalability of Dijkstra's Algorithm with delay constraints for finding multiple shortest disjoint paths in a given network. (ii). Modifying MDPAlg to include a delay threshold to generate multiple shortest disjoint paths constrained by delay. (iii). Comparing the above two approaches in a comprehensive manner to judge when and where they are useful.

## II. Algorithm

### A. Delay Bounded Dijkstra's Algorithm

The shortest path (SP) problems are core networking problems. In practical scenarios, the paths with a total cost greater than a given bound are not used. As Dijkstra's algorithm discovers paths in increasing order of cost, the SP search can be terminated earlier, thereby reducing the search space and runtime of the algorithm [1].

Additionally, a delay constraint is introduced to the algorithm, where accumulated delays from the source to other nodes are tracked in the same way that costs are calculated in the simple Dijkstra algorithm. When moving from one node to the next, if the delay exceeds the threshold, the corresponding cost is not updated. This allows the algorithm to terminate early if the delay constraint is not met, reducing the search space and runtime of the algorithm. By incorporating delay constraints into the algorithm, it is possible to find the shortest path that meets both the cost and delay requirements. One approach to generate multiple paths using Dijkstra's algorithm is to run it repeatedly on the modified graph, where the links which are used in the previous output path are removed.

Edge disjoint shortest path algorithm proposed in [2] calculates the shortest path using an iteration of Dijkstra, then the graph is modified by switching the direction of the links present in this shortest path and multiplying their weight by -1 so that these links are not considered when Dijkstra is run again, resulting in a pair of disjoint shortest paths. A similar approach is followed in Suurballe's algorithm, [3].

The iterative use of Dijkstra's has its disadvantages. Firstly, running Dijkstra's algorithm repeatedly on the modified graph can significantly increase the runtime, particularly for large and complex networks. This can have a negative impact on the overall performance of the network, particularly in time-sensitive applications. Secondly, this approach may not always find all the shortest paths that exist in the network. This is because when the links of a discovered shortest path are removed from the graph, it is possible that some of the shortest paths, which may exist in the original graph, may not be discovered in subsequent runs of Dijkstra's algorithm.

### *B. Delay Modifications to MDPAlg*

In order to increase the reliability of the network, multiple paths are needed so that the same data can be sent simultaneously over different paths. The algorithm described in Section II-A has the limitation of producing only one path at a time, which can be a significant drawback in applications where network reliability is critical.

To overcome these issues, Lopez-Pajares et al. [5], designed MDPAlg which is able to obtain multiple disjoint paths among a given node and the remaining nodes in a graph, following a two-phase process, with just a single full graph search. During the first phase, the aggregated costs to go from the source to any other node in the graph are determined. This analysis is a modified version of Dijkstra's algorithm in which a graph node not only collects information: delay and cost, from the minimum-cost tree but from all its neighbors (including cross-links), lines 9-11 in Algorithm 1. By using this extra information, MDPAlg is able to build multiple disjoint paths with only one single graph search, hence avoiding iterative executions.

The Cost Matrix (CM) is defined as an NxN non-complete matrix of depth 2, where N is the total number of nodes in the graph. The matrix is non-complete because if two nodes don't share a link between them, the corresponding cell in the matrix will be empty. Algorithm 1 illustrates the analysis phase. We initialize CM with an infinite value in all of its entries, except for the one associated with the source, set up with zero cost and delay, which ensures that the algorithm starts the analysis procedure at this node. Then each node of the graph is analyzed once, ordered by the selection criteria of the lowest cost, see line 4 in Algorithm 1.

The first depth of CM corresponds to the cost (or weight) and the second corresponds to the delay values and both are filled respectively in line 9-14.

The path generation phase is based on the CM obtained in Algorithm 1. Path generation follows an iterative approach of getting the minimum cost neighbor and disabling the links which are used. More specifically, after selecting a neighbor node to continue the path construction, the two entries representing the link between them in the CM are disabled (since the paths obtained are bidirectional). We modify this matrix with values calculated in a delay matrix. Specifically, for every value in the delay matrix which exceeds the delay bound, the corresponding value in CM is replaced by infinity, therefore in the path generation phase, this link will never be selected.

**Algorithm 1:** Cost Analysis in MDPAlg

```
   Data: Graph(G), Source(s)
   Result: Cost Matrix(CM)
1  CM = initialize(G, s);
2  Q = get_nodes_from G;
3  while Q ≠ ∅ do
4      u = extract_min(Q);
5      for each vertex v neighbour of u do
6          if v ≠ s then
7              w = L_c ;            /* Link Cost */
8              d = L_d ;            /* Link Delay */
9              if CM[u][u][0] + w < CM[v][v][0] then
10                 CM[v][v][0] = CM[u][u][0] + w
                   CM[u][v][0] = CM[u][u][0] + w
11             end
12             if CM[u][u][1] + d < CM[v][v][1] then
13                 CM[v][v][1] = CM[u][u][1] + d
                   CM[u][v][1] = CM[u][u][1] + d
14             end
15         end
16     end
17 end
18 return CM
```

### *C. Delay Constraints*

Delay constraints for real-time communication have received significant research attention over the last few years [6, 7]. Determining delay bounds on a per-hop basis is a non-trivial task, as they also depend on the current reservation of flows in the network and their paths. The proposed routing algorithms are run iteratively for each flow in the network. However, when additional flows are added to the network, previous delay constraints may become invalid. Consequently, previously determined paths could be no longer valid. Therefore, in [8], we propose a central network controller which offers reservation-independent delay bounds for IEEE 802.1Q networks [9]. The controller defines maximum delay bounds for each hop in the network, which are then used for the routing algorithms. These delay bounds are validated for each new flow reservation, to make sure that all delay constraints - and returned paths - remain valid. We use Network Calculus [10] - as it is a well-established delay analysis framework - to validate the per-hop delay bounds when new flows are added to the network. An overview of related work for the delay analysis of IEEE 802.1Q networks using Network Calculus can be found in [6]. Be aware that the simultaneous use of disjoint paths to duplicate the transmission of frames, as proposed by the Time-Sensitive Networking standard IEEE 802.1CB [11] (Frame Replication and Elimination for Reliability), increases the delay in the network, as has been identified in [12, 13].

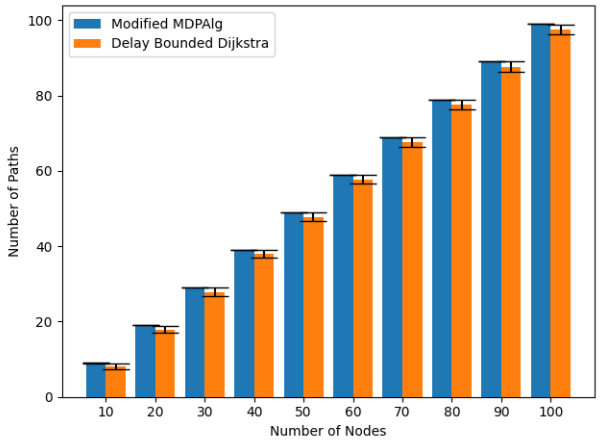


Fig. 1. Number of paths generated

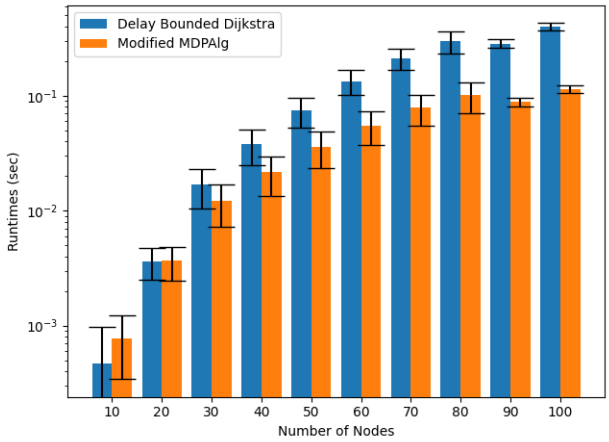


Fig. 2. Run time comparison

## III. Results

The modified algorithms were first implemented in Python. Our modified MDPAlg is compared with Delay Bounded Dijkstra's because it is a well-proven solution. However, as Dijkstra's algorithm does not provide disjoint paths by definition, we must execute Dijkstra's algorithm iteratively to obtain the disjoint paths between a given pair of nodes. After each run, we remove from the graph the minimum-cost path obtained, launching Dijkstra's algorithm with the modified graph again to calculate a new disjoint path. The process ends when no more paths are discovered between the pair of nodes. In this way, the disjointness among paths is guaranteed.

The experiment involves testing on the most challenging type of graphs, which are fully connected, and contain between 10 and 100 nodes. For each instance, 1000 test runs are conducted, with randomly assigned weights and delays for evaluation. Moreover, the source-destination pair is randomly chosen in those 1000 runs. From Fig. 1, we see that MDPAlg manages to return more paths than Dijkstra's. This is because the graph changes every time Dijkstra's is executed, it is not able to find the maximum number of paths.

The comparison of run-times is illustrated in Fig. 2, and it is evident that the primary factor causing the algorithm to slow down is the execution of Dijkstra's. MDPAlg, on the other hand, solves this issue by conducting a single graph search only once, which makes the algorithm faster. Additionally, as MDPAlg is capable of generating multiple disjoint paths, it is beneficial to use our modification to eliminate paths that exceed the delay threshold. This approach performs better than the traditional approach of using Dijkstra's algorithm.

## IV. Conclusion

This paper proposes modifications to MDPAlg to include a delay threshold to generate multiple shortest disjoint paths constrained by delay, making them more scalable and efficient in larger networks. The contributions of this paper include evaluating the efficiency and scalability of Dijkstra's Algorithm with delay constraints for finding multiple shortest disjoint paths in a given network, modifying MDPAlg to include a delay threshold to generate multiple shortest disjoint paths constrained by delay, and comparing the above approaches in a comprehensive manner.

In summary, the algorithm described in this paper provides a solution to the routing problem in computer networks by determining the most efficient path for data packets to travel from one node to another while considering the cost of the path and its delay. Incorporating delay constraints into the algorithm ensures that the shortest path that meets both the cost and delay requirements is found, making the network operate efficiently and effectively. The proposed modifications to Dijkstra's algorithm and MDPAlg make them more scalable and efficient, allowing for the generation of multiple disjoint paths in larger and more complex networks.

## References

[1] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik*, vol. 1, pp. 269–271, 1959.

[2] R. Bhandari, "Optimal physical diversity algorithms and survivable networks," in *Proceedings Second IEEE Symposium on Computer and Communications*, pp. 433–441, 1997.

[3] J. W. Suurballe and R. E. Tarjan, "A quick method for finding shortest pairs of disjoint paths," *Networks*, vol. 14, no. 2, pp. 325–336, 1984.

[4] A. V. Bemten, J. W. Guck, C. M. Machuca, and W. Kellerer, "Bounded dijkstra (bd): Search space reduction for expediting shortest path subroutines," *ArXiv*, vol. abs/1903.00436, 2019.

[5] D. Lopez-Pajares, E. Rojas, J. A. Carral, I. Martinez-Yelmo, and J. Alvarez-Horcajo, "The disjoint multipath challenge: Multiple disjoint paths guaranteeing scalability," *IEEE Access*, vol. 9, pp. 74422–74436, 2021.

[6] L. Maile, K.-S. Hielscher, and R. German, "Network Calculus Results for TSN: An Introduction," in *IEEE Information Communication Technologies Conference (ICTC)*, (Nanjing, China), pp. 131–140, May 2020.

[7] L. Deng, G. Xie, H. Liu, Y. Han, R. Li, and K. Li, "A survey of real-time ethernet modeling and design methodologies: From avb to tsn," *ACM Comput. Surv.*, vol. 55, no. 2, 2022.

[8] L. Maile, K.-S. J. Hielscher, and R. German, "Delay-guaranteeing admission control for time-sensitive networking using the credit-based shaper," *IEEE Open Journal of the Communications Society*, vol. 3, pp. 1834–1852, 2022.

[9] "IEEE Standard for Local and Metropolitan Area Network–Bridges and Bridged Networks," *IEEE Std 802.1Q-2018 (Revision of IEEE Std 802.1Q-2014)*, 2018.

[10] J.-Y. Le Boudec and P. Thiran, *Network calculus: A theory of deterministic queuing systems for the Internet*. Berlin, Heidelberg: Springer-Verlag, 2001.

[11] "IEEE standard for local and metropolitan area networks - frame replication and elimination for reliability," *IEEE Std 802.1CB-2017*, 2017.

[12] L. Maile, D. Voitlein, K.-S. Hielscher, and R. German, "Ensuring reliable and predictable behavior of ieee 802.1cb frame replication and elimination," in *ICC 2022 - IEEE International Conference on Communications*, pp. 2706–2712, 2022.

[13] L. Thomas, A. Mifdaoui, and J.-Y. Le Boudec, "Worst-case delay bounds in time-sensitive networks with packet replication and elimination," *IEEE/ACM Transactions on Networking*, vol. 30, no. 6, pp. 2701–2715, 2022.