Globally balanced paths for minimizing congestion in HPC networks

Jan De Neve*
Ghent University - imec
Ghent, Belgium
janf.deneve@ugent.be

Wouter Tavernier
Ghent University - imec
Ghent, Belgium
wouter.tavernier@ugent.be

Didier Colle
Ghent University - imec
Ghent, Belgium
didier.colle@ugent.be

Mario Pickavet
Ghent University - imec
Ghent, Belgium
mario.pickavet@ugent.be

Abstract

Network bandwidth is a major bottleneck for the performance of HPC workloads. Load balancing techniques are required that distribute traffic over multiple parallel paths to avoid that some links become overly congested. In this work, we focus on determining the paths between every node pair in the network such that the links of the network are used as uniformly as possible. These paths then facilitate an adequate load balancing.

We propose a problem formulation for the optimal set of paths and also introduce an efficient algorithm that approximates these optimal paths. We analyze how well this heuristic algorithm performs at balancing the link usage in the network and demonstrate that a very uniform link usage can be obtained. We also show how these balanced paths reduce the congestion of the links in HPC networks. This analysis is done for networks of different types and sizes, assuming basic and advanced load balancing techniques.

Keywords

Congestion minimization, HPC network, routing

1 Introduction

In recent years, the computing power deployed in data centers has increased immensely. This has allowed the emergence of AI models trained on huge data sets. As a result, the network bandwidth has become a major bottleneck in High-Performance Computing (HPC) clusters [2, 16].

Traditionally, HPC clusters often have a Clos-based [8] network topology. More recently, topologies with lower diameter have been proposed which are more cost-effective and reduce the latency in the network. Examples of such topologies are Dragonfly [11], Slimfly [4], Polarfly [12], which asymptotically approaches the Moore bound [9], and Jellyfish [13], which uses random graphs for its structure.

To balance the traffic load in the HPC network and avoid congested links, traffic from a source node i to a destination node j is sent over multiple parallel paths, via flow splitting [1] or packet spraying. How to deal with the technological implications of packet spraying is described in the recent Ultra Ethernet standard [14]. An SDN switch stores the different parallel paths in its flow table.

*Corresponding author

INOC '26, Liège (Belgium)

A problem with the parallel paths in low-diameter topologies is that there is often only one or a few shortest paths between a pair of nodes [3]. This makes the popular routing strategy Equal Cost Multipath (ECMP) [10] unsuited for these topologies, as it only spreads traffic over shortest paths. Therefore, routing strategies must be considered that also use the almost-shortest paths in the network.

When the paths are determined between all node pairs, they are used as input for a load balancing algorithm. The load balancing decides how much traffic from node i to node j is sent over the different available paths from i to j. Popular load balancing mechanisms like ECMP or UGAL [7] use local decisions, while we consider load balancing based on global decisions. In this context, global means that we use information on the entire state of the network.

In this work, we investigate ways to determine multiple parallel paths between each node pair in the network. We keep track of how often each link in the network has been used by different paths and try to balance the global link usage as well as possible. In Section 2, the network model is defined and the problem statement is introduced of what are considered to be optimal paths. In Section 3, we define a baseline algorithm and improve upon this baseline to obtain pathfinding algorithms that approximate the optimal solution of the problem defined in Section 2. In Section 4, we analyze how well the pathfinding algorithms can balance the paths in the network and also look how these balanced paths can improve the congestion in the HPC network.

2 Problem definition

2.1 Network model

We model the HPC network as a graph $G = (V, E)$. Each node $v \in V$ represents one switch in the network. Each edge represents a direct link between two nodes in the inter-switch network. We consider directed links, so an edge $\{u, v\}$ means that there is a link from node u to node v and also a link from node v to node u. Note that the switches are also connected to the endpoints/servers in a rack, typically ~40 endpoints per rack, but we do not consider these individual endpoints. In our model, the traffic from the endpoints is aggregated in the switch and we are only concerned with the traffic in the inter-switch network, so not the traffic between endpoint and switch. We call the number of nodes in the graph $n = |V|$ and the number of links $m = 2|E|$.

Each node pair $[i, j]$ has a collection of k edge-disjoint paths from i to j associated with it. Traffic from node i to node j is sent over (some of) these k paths, according to weights $w_{i,j,x}$. The weight $w_{i,j,x}$ corresponds to the fraction of the traffic between node i and node j that is sent over the x^{th} path out of the k

10.48786/inoc.2026.15

possible paths. All weights corresponding to one node pair must therefore sum up to one: $\sum_{x=1}^{k} w_{i,j,x} = 1, \forall i, j$. The distribution of these weights is crucial for adequate load balancing in the system.

The focus of this research is to determine k edge disjoint paths for every node pair, such that the usage of the links in the network is spread along all the paths as uniformly as possible. An algorithm that finds such $kn(n-1)$ valid paths is called a pathfinding algorithm. The more uniformly the links in the network are used, the less likely it is that any link becomes heavily congested when traffic is sent over all the paths. Note that the pathfinding algorithms have no knowledge about traffic in the network, they only consider the topology.

2.2 Optimal path selection

To enforce a uniform distribution of the links, we introduce a metric called path count of a link $[u, v]$, defined as $P_{u,v} = \sum_{i,j \in V} \sum_{x \leq k} \Gamma_{u,v}^{i,j,x}$, where $\Gamma_{u,v}^{i,j,x}$ is 1 if link $[u, v]$ occurs in the x^{th} path of node pair $[i, j]$, and is 0 if not. A link with a high path count is more likely to be congested, so for an optimal load balancing in the system, we want to minimize the maximal path count over all links in the network.

The value of k is a trade-off that must be made. More parallel paths means more options to send traffic over different paths and therefore a better chance of balancing the load in the network. If k is smaller, a load balancing algorithm that determines the weights of the paths will work faster due to the smaller number of paths it has to take into account. Another advantage of a small k is that fewer weights have to be updated in the Network Interface Cards (NICs) every reconfiguration period. This results in a lower configuration time and the network can therefore react more quickly to changes in the network traffic. Moreover, it has been shown that a small number of parallel paths can be sufficient for load balancing in an HPC network [5]. The average path length, and therefore the latency, will also increase with higher k, but this effect is minimal because a decent load balancing algorithm will generally give more weight to shorter paths.

There is typically much freedom to determine the k paths, although they have to adhere to some constraints. First, they must be edge-disjoint. The idea behind sending traffic from node i to node j over multiple paths is that the intermediate links could become too congested if only a single path is used. However, if multiple paths share one or more links, these common links are still at higher risk of becoming congested, defeating the point of having multiple paths. Second, the paths must use as few hops as possible. On the one hand, longer paths have a higher latency, which might negatively impact the performance of the system. On the other hand, if the paths use more links, the potential for congestion in the network increases. The k shortest paths between a given node pair $[i, j]$ can be obtained with a basic single-commodity minimum-cost flow (MCF) problem, where a flow of k needs to go from node i to node j and each link has capacity 1 and cost 1. The capacity 1 guarantees that the paths are edge-disjoint. The constant cost 1 guarantees that the total number of links used in the k paths is minimal for this node pair.

The MCF problem described above gives guarantees for edge-disjointness and shortest path lengths locally for every node pair. If we just solve $n(n-1)$ MCF problems independently for each node pair, we get valid paths we can use for the load balancing in the HPC network. However, we have no guarantees on how often each link is used by different paths and how uniform the total link usage will be. A globally uniform link usage is exactly what we want to obtain in this research, so we need a more global problem formulation than these $n(n-1)$ independent MCF problems. We could try to model the problem as a multi-commodity MCF problem, where all $n(n-1)$ node pairs have a flow k that needs to be routed. However, to allow a link to be used by paths of different node pairs, its capacity needs to increase. In this case, one node pair could also use this extra capacity to send more than one path through a certain link, which would violate the edge-disjointness of paths belonging to one node pair. Additional constraints would need to fix this issue, which would make the problem more complex than a standard multi-commodity MCF problem.

We introduce the Globally Balanced All-to-All Min Cost Flow (GBAMCF) problem. In this formulation, we combine the $n(n-1)$ MCF problems of every node pair, but make them into constraints rather than optimization problems themselves. The model has $n(n-1)m$ binary variables, where $x_{i,j,e}$ is 1 if the link e is used in one of the k paths of node pair $[i, j]$ and it is 0 if it is not used by this node pair.

The $n^2(n-1)$ constraints (2) are the flow constraints which can also be found in a standard MCF problem. The set of edges arriving in a node u is denoted by $I(u)$ and the set of edges departing from u is denoted by $I^*(u)$. The function $\delta_{x,y}$ equals 1 if $x = y$, else 0. These flow constraints thus make sure, for a given node pair $[i, j]$, that each intermediate node u has as much flow arriving into it as there is flow departing from it, giving a net zero flow. For the source node i, a net flow of k is departing and for the destination node j, a net flow of k is arriving.

The $n(n-1)$ constraints (3) correspond to what is originally the objective function of each individual MCF problem. However, here we assume that the individual MCF problem has been solved for each node pair $[i, j]$ and that the optimal value is stored in $\text{MCF}_{i,j}$. By making these individual objective functions into constraints where they have to equal their respective optimal solutions, we enforce that the k paths for any node pair $[i, j]$ correspond to some optimal solution of the MCF problem, and therefore use the minimum required number of links. Note that without these constraints, the objective value P_{max} might be improved. However, we impose the minimal path lengths to guarantee a low latency in the network, which is not reflected in the objective value, but still important in real systems.

The m constraints (4) define the meaning of the variable P_{max}, the maximal path count in the network. In these constraints, the number of paths in which a link e occurs are counted. This could be defined as P_e, the path count of the link e, but instead we leave the definition of P_e implicit and immediately enforce that the maximal path count P_{max} should be greater than or equal to every P_e, de facto setting it equal to the maximal P_e. When P_{max} is minimized, we have the optimal solution to the GBAMCF problem. The set of directed links in the network is denoted as E^*.

Objective function:

$$\text{Minimize: } P_{max} \tag{1}$$

Constraints:

$$\sum_{e \in I(u)} x_{i,j,e} - \sum_{e \in I^*(u)} x_{i,j,e} = k(\delta_{u,j} - \delta_{u,i}) \quad \forall i, j, u \in V : i \neq j \tag{2}$$

$$\sum_{e \in E^*} x_{i,j,e} = \text{MCF}_{i,j} \quad \forall i, j \in V : i \neq j \tag{3}$$

$$\sum_{i,j \in V : i \neq j} x_{i,j,e} \leq P_{max} \quad \forall e \in E^* \tag{4}$$

$$x_{i,j,e} \in \{0,1\} \quad \forall i, j \in V : i \neq j, \forall e \in E^* \tag{5}$$

3 Pathfinding algorithm

Because of the high number of variables and constraints, the GBAMCF problem becomes infeasible to solve optimally even for small networks. We therefore introduce a heuristic algorithm that gives a valid solution in reasonable time. We start with a baseline algorithm that gives a valid but low quality solution. We then introduce different ways to improve the solution until it converges to a high quality solution with highly balanced path counts.

For the algorithm, we revisit the idea of the k paths between a node pair $[i, j]$ being the solution to a single MCF problem. We define $S_{i,j}$ as the set of edges making up the solution of the MCF problem that finds the k paths between a node pair $[i, j]$. In the baseline, we solve all $n(n-1)$ MCF problems independently. In a solution using the baseline algorithm, it is possible that some link e_1 is used by some path in almost all node pairs, while some other link e_m is used by very few node pairs. This typically happens because the link e_1 has a lower index in the adjacency lists of the network graph. During the MCF algorithm, the lowest indexed link is chosen in case of a tie. This baseline algorithm is called `multi-MCF`.

The problem where the algorithm is biased to prefer links with a lower index is easily mitigated by introducing randomness. This means that a random link is chosen in case of a tie, instead of the lowest indexed link. Although we can now statistically avoid the extremely high path counts for the links with a low index, we still have no explicit way to keep a global balance in the path count. This extension of the baseline algorithm is called `multi-MCF-random`.

For further improvement, we no longer treat the $n(n-1)$ MCF problems as independent problems. When solving the MCF problem for some node pair $[i, j]$, we replace the constant link costs by dynamic link costs. Each link now gets a cost $1 + P_e\delta$, where P_e is the path count of the link for all paths of other node pairs that have already been determined, and δ is a small constant. The value of δ should be sufficiently low so that $\sum_{e \in S_{i,j}} P_e\delta$ is always smaller than 1, otherwise we lose the guarantee that this MCF problem returns paths with a minimal total length. For example, $\delta = \frac{1}{n^2 m}$ would be fine, as there are only $n(n-1)$ node pairs so $P_e \leq n(n-1) < n^2$ for every link, and $|S_{i,j}|$ is upper bounded by the number of links in the network m as we only have edge-disjoint paths. In conclusion, $\sum_{e \in S_{i,j}} P_e\delta = \sum_{e \in S_{i,j}} \frac{P_e}{n^2 m} < 1$, so we keep the guarantee of having the shortest paths in the solution to each individual MCF problem. By making link costs higher for links that are already used a lot by paths in previous node pairs, we force the MCF algorithm to prefer links that are not yet being used a lot. This algorithm is called `multi-MCF-penal`, because the heavily used links are penalized.

To finalize the pathfinding algorithm, we make better use of the dynamic link costs. The main problem with the current situation is that the dynamic costs only give valuable information when a significant amount of paths have already been determined. At the start of the algorithm, there is no information, so the first selected paths might be far from optimal. This can be solved by making our algorithm cyclic. When the algorithm has found all $kn(n-1)$ paths for the entire network, we just start over with all the information we currently have. We delete the k paths we had before between nodes i and j and determine k new paths for this node pair, using the dynamic link costs based on the entire solution we currently have. When this process is repeated for many cycles, the solution converges to a high quality solution where the path counts of all links in the network are very balanced. This algorithm is called `multi-MCF-cyclic`.

The heuristic pathfinding algorithm is efficient because it only needs to solve successive MCF problems. An MCF problem can be solved quickly, for example using the Busacker-Gowen algorithm [6] with a worst-case time complexity of $O(knm)$. An MCF problem must be solved for all $n(n-1)$ node pairs, giving a total worst-case time complexity of $O(kn^3m)$ for the first three algorithms, and $O(ckn^3m)$ for `multi-MCF-cyclic` if c cycles are done.

4 Analysis

In this section, we analyze the performance of our pathfinding algorithm for the GBAMCF problem. We focus on the Dragonfly network with 510 nodes, abbreviated to DF510, for our elaborate analysis. Keeping in mind the benefits of a low k presented in Section 2.2, we fix the number of parallel paths between each node pair to $k = 4$. In the final subsection, we look at the implications of the balanced paths for the actual congestion in the network. In this flow-level analysis we consider networks of different types and sizes.

4.1 Balancing path counts

For the DF510 network, the runtime of one pathfinding algorithm (or one cycle in `multi-MCF-cyclic`) is 10 minutes (C++ implementation, g++ compiler with -Ofast flag, executed on an Intel E5520 2.2GHz CPU). The paths are precomputed, so this is not a time-critical step. Still, the pathfinding algorithms have potential for parallelization to decrease the runtime for larger networks.

Figure 1 shows the distribution of path counts for the links in the DF510 network. If a link has a path count 1000, there will be a bar at the value 1000 on the x-axis and the more links that have path count exactly 1000, the higher this bar will be. Instead of showing the number of links that have this exact path count on the y-axis, the relative contribution is shown of links with this path count. This means that if one link has path count 1000, its bar will be ten times higher than the bar of another link that has path count 100. This is done to normalize the plots, so that they all add up to one if the bars of one plot were added. Either way, these plots show for each algorithm which paths counts are present in the solution and if so, how often they occur.

When using the baseline algorithm `multi-MCF`, it can be seen that the path counts are very unbalanced. Some links have a path count of more than 3000, while other links are used by fewer than 100 node pairs. These links with the highest path count are at severe risk of getting congested if the paths of this solution are used to route traffic in an HPC network, making it a bottleneck in the system. The links with the lowest path count will likely be underused, which is a waste of resources.

Adding randomness to the baseline algorithm already improves the situation a lot, as can be seen in the plot for `multi-MCF-random`. Every link now has a path count of less than 1000. There is a high peak of links that have a path count just below 1000 so there are many potential links which can form a bottleneck when these paths are used for routing.

Finally, we consider `multi-MCF-penal`, the algorithm that penalizes the overly used links. There is no longer a peak near

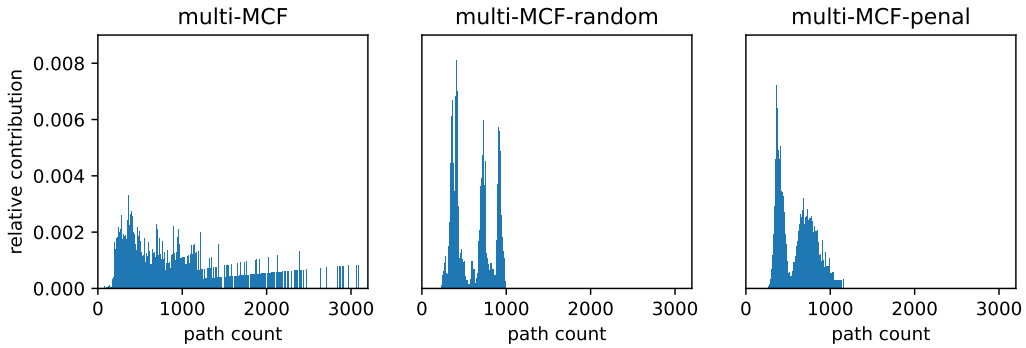

Figure 1: Distribution of path counts for the pathfinding algorithms `multi-MCF`, `multi-MCF-random`, and `multi-MCF-penal`.

1000, so fewer links are at risk of forming a bottleneck. However, the highest path count P_{max} is now 1153, which is higher than for `multi-MCF-random`. As stated in the previous section, `multi-MCF-penal` only performs well when many paths have already been determined. The high maximal path count is due to many suboptimal decisions being made in the early stages of the algorithm, when little information is known, so the dynamic link costs $1 + P_e\delta$ are not yet effective. Exactly this problem is what we solve with the final algorithm, `multi-MCF-cyclic`.

Figure 2 shows the distribution of path counts for the links in the same DF510 network, but now for `multi-MCF-cyclic`. Each plot shows the situation after c cycles are completed. This means that the plot for $c = 1$ shows the same data as the plot for `multi-MCF-penal` in Figure 1, as these algorithms are exactly the same if only one cycle is done.

Note that we now make a distinction between two types of links, intra-group links and inter-group links. This is a property inherent to the Dragonfly network. The 510 nodes in the DF510 network are placed in 51 groups of 10 nodes. The 10 nodes in one group are fully interconnected, meaning that they all have a direct (intra-group) link to every other node in the group. All the other links in the network are inter-group, connecting two nodes of different groups and thus ensuring full connectivity throughout the network. We did not explicitly consider this distinction in link type when conducting our experiments, but it automatically appeared in the form of two separate peaks in the histogram. To confirm that the two link types in the network correspond to the peaks in the histogram, we gave them a different color. It can be seen that the algorithm naturally separates the intra-group links from the inter-group links, even though it just takes a graph as input, without any information on different link types.

From the solution it can be seen that the inter-group links are used more often than the intra-group links and are therefore likely to be more congested. This can be dealt with by clever job placement, meaning that jobs that communicate heavily are placed on endpoints whose switches are in the same group [15]. The HPC network can also be designed such that the inter-group links are links with a higher bandwidth than the intra-group links, because more traffic is expected to be routed over them.

As the number of cycles increases, the path count distribution converges to two separated peaks. This can already be seen after 1 cycle, although the peaks are not very high and there is still overlap. After 2 cycles, the solution quality has improved a lot and it keeps improving as more cycles are done. After more than 20 cycles, the algorithm has almost converged and it finally ends with a solution where P_{max} equals 715. The height and narrowness of the peaks show that `multi-MCF-cyclic` performs well at balancing the path counts and minimizing P_{max}, therefore making it a suited heuristic algorithm for the GBAMCF problem. The mathematical objective of minimizing P_{max} of course originates from the problem of load balancing in the HPC network. In the next subsection, we investigate how the balanced paths can improve the congestion in the HPC network in different scenarios.

4.2 Congestion minimization

The reason why nodes i and j have k different paths between them, is to avoid congested links as much as possible. If the total amount of traffic in the network is low and there are no congested links, the traffic from node i to node j will just be sent over the shortest path(s) out of the k possible paths. However, if some link e is already congested due to a high traffic load of other nodes in the network, node i will not send its traffic to node j over a path that contains link e, even if this path is the shortest. Which fraction of traffic from node i to node j is sent over which of the k paths is determined by the weights $w_{i,j,x}$. These $kn(n-1)$ weights in total are crucial for minimizing congestion in the network.

We consider two ways of determining $w_{i,j,x}$. The first scenario is a basic uniform weight distribution. In this setting, all weights are equal to $\frac{1}{k}$, thus conforming to the constraint $\sum_{x=1}^{k} w_{i,j,x} = 1$, $\forall i, j$. The advantage of this technique is its simplicity. The network strives for balance by evenly spreading out the traffic over the k paths for every node pair. No overhead goes to the reconfiguration of the switches in the network when the weights are updated, as the weights always stay the same. However, by remaining constant, the weights are oblivious to the traffic demand in the network. Note that this scenario is similar to ECMP routing, but now the traffic is evenly spread over the k predetermined paths, some of which are not necessarily shortest paths, while ECMP spreads the traffic only over shortest paths.

In the second scenario, we choose the optimal weights for minimizing the congestion in the network. For these weights, an optimization problem must be solved where we minimize the congestion of the most congested link in the network. This optimization problem is specific to the $kn(n-1)$ paths determined by the pathfinding algorithm, as well as the specific traffic demands between all $n(n-1)$ pairs of nodes. The performance of this technique is evidently better than in the first scenario, as it is optimal for the given paths and traffic demands, but this comes

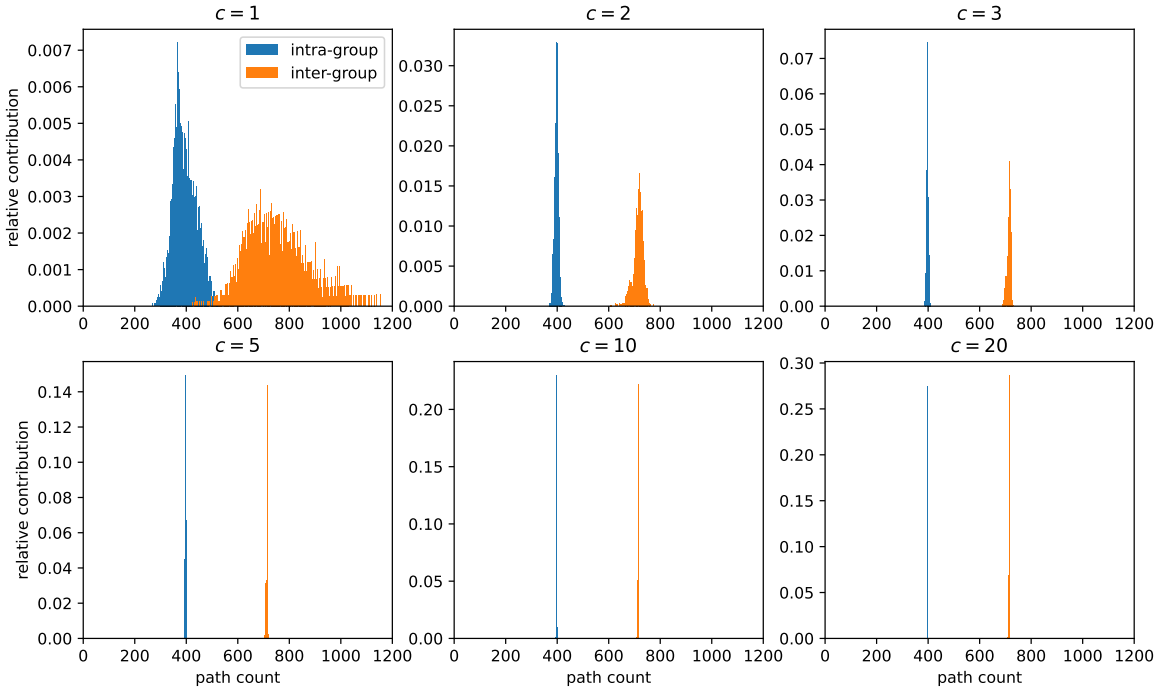

Figure 2: Distribution of path counts for `multi-MCF-cyclic` after different amounts of cycles c.

at a high computational cost. For large networks, the optimization problem for determining the weights takes a long time to solve. If the traffic demands in the HPC network change more quickly than the weights can be calculated, they will no longer be optimal.

These two scenarios show two extremes, the uniform method being very basic and the optimal method being the solution of a complex problem. In practice the solution will be somewhere in between, with suboptimal weights that can be calculated efficiently and take into account the traffic demands as well as possible. Such methods are out of scope for this paper.

The goal of this subsection is to investigate to which extent paths that score better on the GBAMCF problem, also result in a lower congestion in the network. We focus on paths given by the pathfinding algorithms from Section 3. We use the two scenarios described above to obtain weights $w_{i,j,x}$ and use these weights for the congestion analysis. Networks of different types and sizes are considered. For each network, we generate a random traffic demand, where the traffic sent from a node i to a node j is a uniform random integer between 0 and 7. This value represents the bandwidth required for the traffic. The exact unit is not relevant, as we only compare the values relative to each other within one network. Table 1 shows the load of the most congested link in the network for the different parameters.

We first focus on the congestion in the DF510 network, as we can easily link these conclusions to the results from Section 4.1. For uniform path weights, the relative congestion performance of the different pathfinding algorithms is highly similar to that of their respective maximal path counts. This shows that the maximal path count, which is optimized in the GBAMCF problem, is a good metric for predicting the congestion in a network, at least under the common uniform random traffic pattern. For optimal path weights, `multi-MCF-cyclic` still performs best, but the difference in congestion is now only a few percent compared to the baseline. This shows that the optimal path weights can to a great extent cope with the imbalance of link usage by the paths and thus result in a congestion that is almost equal to the congestion under balanced paths. We also observe that `multi-MCF-penal` has a better congestion for optimal path weights than `multi-MCF-random`, while it was the other way around for the uniform path weights. A possible explanation for this is the high peak of path counts just under 1000 for `multi-MCF-random`, while `multi-MCF-penal` has fewer links with such a high path count, despite the maximal path count being higher. It is thus easier for the optimal path weights to mitigate congestion issues for the few links with a high path count in `multi-MCF-penal` than for the many links with a high path count in `multi-MCF-random`.

The same analysis is done for the Dragonfly network with 114 nodes (DF114), the Polarfly network with 307 nodes (PF307), and the Slimfly network with 722 nodes (SF722). Again, the performance of the pathfinding algorithms regarding maximal path count is well reflected in the uniform path weights. For optimal path weights, the difference in congestion for the non-baseline algorithms is even smaller compared to the DF510 network, being less than 0.1% in the PF307 and SF722 network.

5 Conclusion

In this paper, we propose the GBAMCF problem for obtaining balanced paths for routing in an HPC network. We construct pathfinding algorithms and show that these algorithms give a solution that performs well on the GBAMCF problem. Our experiments show that the balanced paths have the potential to decrease the congestion in an HPC network where a weight is given to each path. For the load balancing in a real-world HPC network, the quality of the path weights will be somewhere in between the quality of the uniform and the optimal path weights. The optimal path weights are not realistic to determine in a short time, while any decent load balancing algorithm should be able

Table 1: Load of the most congested link in the network

Network	DF114		PF307		DF510		SF722	
Path weights	uniform	optimal	uniform	optimal	uniform	optimal	uniform	optimal
`multi-MCF`	535.0	136.14	821.5	120.75	2721.75	370.55	1635.25	193.54
`multi-MCF-random`	302.25	135.59	217.25	119.48	895.5	367.89	319.75	192.94
`multi-MCF-penal`	349.0	134.23	239.0	119.46	987.25	363.37	339.25	192.96
`multi-MCF-cyclic` (20 cycles)	256.75	133.61	190.25	119.46	679.5	360.57	275.25	192.94

to outperform the basic uniform scenario. The impact of the pathfinding algorithms on the network congestion is expected to inversely follow the quality of the path weights: a high impact for low quality, close-to-uniform path weights and a lower, but non-negligible impact for high quality, close-to-optimal weights. The more dynamic the traffic in the network is, with quickly changing demands, the less time a load balancing algorithm has to calculate decent path weights, thus the more important the GBAMCF problem and the well-balanced paths become for the routing in the network.

Acknowledgments

This work was supported by the Special Research Fund of Ghent University under grant 01D14623 and by imec's AAA funding project Net4HPC. This work was partly supported by the BOF-BAF project bof/baf/4y/2024/01/806.

References

[1] Vamsi Addanki, Prateesh Goyal, Ilias Marinos, and Stefan Schmid. 2025. Ethereal: Divide and Conquer Network Load Balancing in Large-Scale Distributed Training. doi:10.48550/arXiv.2407.00550 arXiv:2407.00550 [cs].

[2] Vamsi Addanki, Oliver Michel, and Stefan Schmid. 2022. PowerTCP: Pushing the Performance Limits of Datacenter Networks. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*. USENIX Association, Renton, WA, 51–70.

[3] Maciej Besta, Jens Domke, Marcel Schneider, Marek Konieczny, Salvatore Di Girolamo, Timo Schneider, Ankit Singla, and Torsten Hoefler. 2021. High-Performance Routing With Multipathing and Path Diversity in Ethernet and HPC Networks. *IEEE Transactions on Parallel and Distributed Systems* 32, 4 (April 2021), 943–959.

[4] Maciej Besta and Torsten Hoefler. 2014. Slim Fly: A Cost Effective Low-Diameter Network Topology. In *SC14: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, New Orleans, LA, USA, 348–359.

[5] Maciej Besta, Marcel Schneider, Marek Konieczny, Karolina Cynk, Erik Henriksson, Salvatore Di Girolamo, Ankit Singla, and Torsten Hoefler. 2020. FatPaths: routing in supercomputers and data centers when shortest paths fall short. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (Atlanta, Georgia) *(SC '20)*. IEEE Press, Article 27, 18 pages.

[6] P. J. Busacker, R. G.; Gowen. 1960. *A Procedure for Determining a Family of Minimum-Cost Network Flow Patterns*. Technical Report. National Technical Information Service.

[7] Ram Sharan Chaulagain and Xin Yuan. 2024. Enhanced UGAL Routing Schemes for Dragonfly Networks. In *Proceedings of the 38th ACM International Conference on Supercomputing* (Kyoto, Japan) *(ICS '24)*. Association for Computing Machinery, New York, NY, USA, 449–459.

[8] Charles Clos. 1953. A study of non-blocking switching networks. *The Bell System Technical Journal* 32, 2 (1953), 406–424.

[9] Cristina Dalfó. 2019. A survey on the missing Moore graph. *Linear Algebra Appl.* 569 (2019), 1–14.

[10] Christian Hopps. 2000. Analysis of an Equal-Cost Multi-Path Algorithm. RFC 2992. doi:10.17487/RFC2992

[11] John Kim, Wiliam J. Dally, Steve Scott, and Dennis Abts. 2008. Technology-Driven, Highly-Scalable Dragonfly Topology. In *2008 International Symposium on Computer Architecture*. IEEE, Beijing, China, 77–88.

[12] Kartik Lakhotia, Maciej Besta, Laura Monroe, Kelly Isham, Patrick Iff, Torsten Hoefler, and Fabrizio Petrini. 2022. PolarFly: A Cost-Effective and Flexible Low-Diameter Topology. In *SC22: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, Dallas, TX, USA, 1–15.

[13] Ankit Singla, Chi-Yao Hong, Lucian Popa, and P. Brighten Godfrey. 2012. Jellyfish: networking data centers randomly. In *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation* (San Jose, CA) *(NSDI'12)*. USA, 225–238.

[14] Ultra Ethernet Consortium (UEC). 2025. Ultra Ethernet Specification 1.0.1. https://ultraethernet.org/wp-content/uploads/sites/20/2025/10/UE-Specification-1.0.1.pdf.

[15] Van Poucke, Dante and Colle, Didier and Pickavet, Mario and Tavernier, Wouter. 2025. Quantifying the impact of job placement and routing on network efficiency in AI clusters. In *PROCEEDINGS OF THE 2025 2ND WORKSHOP ON NETWORKS FOR AI COMPUTING, SIGCOMM 2025* (Coimbra, Portugal). ACM, 74–80.

[16] Yijia Zhang, Taylor Groves, Brandon Cook, Nicholas J. Wright, and Ayse K. Coskun. 2020. Quantifying the impact of network congestion on application performance and network metrics. In *2020 IEEE International Conference on Cluster Computing (CLUSTER)*. 162–168.