Demand-Aware Multi-Source IP-Multicast: Minimal Congestion via Link Weight Optimization

Matthias Bentert
University of Bergen

Max Franke
TU Berlin

Darya Melnyk
TU Berlin

Arash Pourdamghani
TU Berlin

Stefan Schmid
TU Berlin & Fraunhofer SIT

***Abstract*—Multicast is a fundamental communication primitive that can improve the efficiency of many distributed systems. However, current algorithms to construct multicast trees only consider link capacities and are oblivious to the bandwidth demands of senders. Such demand-oblivious approaches can result in suboptimal resource allocations and congestion.**

In this work, we initiate the study of a demand-aware multi-source IP-multicast. In particular, we consider how an operator can optimize link weights to minimize congestion along multiple (and hence possibly *overlapping*) multicast trees. We show that this problem is NP-hard even in very restricted settings such as (i) where there are only two possible link weight values or (ii) where the graph contains only a single receiver. To obtain optimal solutions as well as a baseline for comparison, we also present a mixed integer linear program. We then suggest two fast heuristics, DA-Picky and DA-Hybrid, based on maximum-bottleneck spanning trees. Our empirical results, based on real-world data, show that our algorithms outperform today's demand-oblivious approach and scale to large networks.

***Index Terms*—Multicast, congestion control, algorithm design**

I. Introduction

IP layer multicast, a key communication primitive, is well studied in theory and practice, and aims to reduce load on both the network components and end-point servers. Its applications range from satellite [1], local/campus [2], to financial [3] networks. In addition, there has been renewed interest in recent years for mass media use cases [4]. This has also led to new standards being developed for multicast routing and tree construction. In particular, BIER [5] is starting to get rolled out by ISPs to replace the more cumbersome PIM [6].

One major aspect that has so far not been considered from either the theoretical nor practical side is that of bandwidth-demand aware multicast. It is common that multicast senders use constant and predetermined bitrates. For example, video live streams could be in either 1080p or 4K. Not only incorporating this information in route selection give senders a guarantee that their required bandwidth has been assigned to them, it would also lead to more efficient utilization of available link capacities. In addition, it would allow to provide guarantees to senders that their required bandwidth is allocated to them.

In this paper, we consider a traffic-engineering perspective on multicast, where IP traffic is forwarded along shortest paths, which can be influenced by operators indirectly, by defining link weights. This model is standard for IP traffic but recently also received more attention in the context of emerging segment routing [7]–[10] and software-defined networks [11]–[13].

In this work we first lay the theoretical foundations, showing that such weight assignment is NP hard. This highlights the need for fast heuristics to tackle this issue. We then introduce two such heuristics and show how they compare to the optimal solution that we derive from a Mixed Integer Linear Program (MILP). By doing so, we show the tangible benefits that bandwidth aware mutlicast can bring to realistic scenarios. We hope this will form the basis for both future research into more efficient heuristics but also encourage the development of protocols that can bring our approach into practice.

A. Our contribution

In this paper, we consider the classic problem of allocating multicast trees in an IP network, where routing follows shortest paths which are determined by link weights. In contrast to prior work, we assume a demand-aware perspective, aiming to improve resource allocations based on the actual traffic pattern by tuning link-weights. We prove that finding optimal link weights for our demand-aware multicast tree allocation problem is computationally hard, i.e., NP-hard, even in very restricted settings such as (i) when we are only allowed to choose among two possible link weights or (ii) when considering only a single receiver. Our results also imply that achieving any finite approximation factor is NP-hard.

We then show how an optimal solution can be obtained using a mixed integer linear program and discuss two fast demand-aware heuristics, DA-Picky and DA-Hybrid (DA stands for demand-aware) that aim to make weight selection more bandwidth-aware. We complement our theoretical results with an empirical evaluation based on both real-world and a variety of synthetic datasets. We find that our novel heuristics can provide better solutions in particular in cases with high bandwidth requirements.

B. Related work

IP traffic engineering is well-studied in the literature [8], [14], [15]. Most of the focus of previous results consider the

This work was supported by the Federal Ministry of Education and Research of Germany in the program “Souverän. Digital. Vernetzt.” Joint project 6G Research and Innovation Cluster (6G-RIC), project identification number: 16KISK030. ISBN 978-3-903176-72-0 © 2025 IFIP.

case where the flows are split in each branching point. In this setting, finding a constant-factor approximation to optimize weights of links for a single source destination pair is NP-hard [14], [16]. These approaches significantly differ from our multicast setting where flows are not splitted.

There are previous works that discuss constructions of a single optimal multicast tree, given various constraints [17], [18]. However, these works do not consider the effects of *overlapping* of such trees, which is a particularly challenging optimization problem. On the other hand, works such as [19], [20] that consider multiple multicast trees, and suggest an optimization approach for it, does not consider the weight setting requirement, and hence their approach does not necessarily follow the shortest weighted paths. Hence, to the best of our knowledge, no prior work has considered a demand-aware congestion optimized *weight setting* in the case of IP multicast for *multiple sources*. The need for such a solution has been further showed in a recent work [21].

The idea of constructing maximum(or minimum)-bottleneck trees has been proposed in prior works with the goal of extending the unicast view, in particular for streaming use cases [22], [23]. The idea of overlapping spanning trees have been used to build optimized overlay networks [24], [25] and also ad hoc networks [26].

From the practical side, the traditional protocol, called PIM [6], that is used for inter-domain multicast routing requires large states on the involved routers. It has therefore partly been replaced by BIER [5], which solves the state issue by encoding multicast routers in an additional header. However, BIER, unlike PIM, does not handle all types of multicast. Instead, it requires extensions that handle the process of determining which routers need to be included in the header.

C. Paper outline

The rest of the paper is structured as follows: In §II we formally introduce the demand-aware (DA) multicast problem. We then show in §III that demand-aware multicast is NP-hard even in very restricted special cases. We introduce our mixed integer linear program in §IV and present heuristics in §V. We then briefly discuss results of our experimental evaluation §VI and conclude our work in §VII.

II. Problem Definition

In this section, we formalize our model and the demand-aware multicast problem, DA-Multicast. Our model captures the case of adjusting link weights with the objective of minimizing the maximum congestion over any link.

We say that a directed graph is symmetric if for each link (u, v) in the graph, the reverse link (v, u) is also present and we only consider symmetric directed graphs. We assume that each link $l \in L$ has a capacity c_l. Let $G = (V, A)$ be a directed symmetric graph where V indicates the set of vertices/nodes (we use these two terms interchangeably). Let $S \subseteq V$ be the set of sources. For each source $s \in S$ we consider an integer b_s as bandwidth demand of the source, and a set R_s as receivers of that source. We aim to optimize variable $C \in \mathbf{N}$ that indicates a positive congestion. In other words, we aim to minimize the links (over)-utilization [1]. The task is to assign weight w_l to each link.This link weight is used later on to determine the path between source and destinations. We consider a full-duplex scenario, where a different amount of flow can be passed over each link. We consider a link weight with the following properties:

- $w_{(u,v)} = w_{(v,u)}$ for each link $(u, v) \in L$,
- there is a unique shortest path[2] $P_{s,r}$ from each source s to each receiver $r \in R_s$, and
- for each link $l = (u, v)$ it holds that

$$\sum_{s\in S}(b_s \cdot \max_{r\in R_s} \mathbf{1}_{l\in P_{s,r}}) \leq c_l + C.$$

Here $\mathbf{1}_{l\in P_{s,r}}$ is one if l belongs to the unique shortest path from s to r and zero otherwise.

Intuitively, we assign weights to all link pairs such that the congestion C is minimized. Each source thereby transfers b_s units via shortest paths to all receivers that subscribed to this source. The data is only transferred once over each link and can be duplicated in the nodes if necessary. You can see an example of the problem, and also an example of assigning weights that can result to a huge congestion in Figure 1.

One of the key reasons that make this model particularly challenging is that we can not split the traffic based on the source or a receiver of a path: all the routing decisions should be done based on the weights. Furthermore, the fact that nodes can duplicate traffic differentiates this problem more from traditional multi-commodity flow problems.

III. Hardness Result

In this section, we show that DA-Multicast is NP-hard, even when considering only two possible integer weights. Before going to the proofs, we need to clarify some preliminaries.

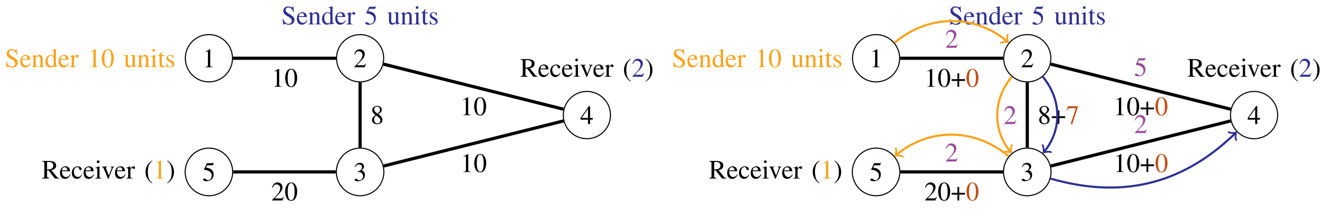
A. Preliminaries

In Vertex-Disjoint Paths (or Edge-Disjoint Paths), we are given an undirected graph G and k terminal pairs $(s_1, t_1), (s_2, t_2), \ldots, (s_k, t_k)$ and the task is to find k paths $P_1, P_2, \ldots, P_k$ such that s_i and t_i are the endpoints of P_i and for each pair $1 \leq i < j \leq k$ it holds that paths P_i and P_j are vertex-disjoint (edge-disjoint), that is, they do not share any vertices (edges). We use the following result due to Middendorf and Pfeiffer [27].

Theorem 1 (Middendorf and Pfeiffer [27], Theorem 2)**.** *Planar Vertex-Disjoint Paths is NP-hard on graphs with maximum degree three.*

[1] Over-utilization of a link is an indication that link capacities need to be updated.

[2] Length of $P_{s,r}$ is the sum of weights w_l along $P_{s,r}$. Focusing on the shortest path ensures compatibility of our model with most traditional IP routing protocols.



(a) An instance of the input to the problem.

(b) Weights, multicast trees and congestion on each link.

Figure 1: Figure 1a shows an instance of the problem with two senders (and their required bandwidth) and two receivers (with their respective sources). The black integers on each line determine the capacity of that link, which is equal for both directions. Figure 1b shows a bad example of setting the weight of each link (in both directions) in purple, and the congestion that link needs to tolerate (given the multicast trees created using the weights) in red. In this case, we can see the congestion of the whole network is 7, which could have been avoided if we set the between nodes 2 and 4 lower than four.

In PLANAR 2P1N-3-SAT, we are given a formula Φ in conjunctive normal form (CNF) where each clause contains two or three literals and each variable appears at most twice positive and exactly once negative in a clause in Φ. Moreover, we are given a planar embedding of the (bipartite and planar) graph H that contains a vertex v_i for each variable x_i and a vertex u_j for each clause C_j and an $\{u_j, v_i\}$ if and only if C_j contains x_i or $\neg x_i$.

Proposition 1 (folklore)**.** PLANAR 2P1N-3-SAT *is NP-hard and, assuming the the Exponential-Time Hypothesis (ETH), it can not be solved in* $2^{o(\sqrt{n+m})}$ *time.*

B. Hardness proof

In this section, we show that DA-MULTICAST is NP-hard even in three very restricted special cases. The first special case is that the input graph is planar, each sender has a bandwidth requirement of 1, and only a single receiver is subscribed to each sender. Moreover, the maximum degree in the graph is six and the question is whether $C = 0$ can be achieved. Note that the latter also rules out any approximation algorithms in terms of C. In the second special case, the graph contains only a single receiver and $C = 0$. In the third special case, we are only allowed to assign weights 1 or 2 to edges and the requirements of the first special case also apply to the third. Our hardness results also imply that DA-MULTICAST can not be solved in $2^{o(n+m)}$ time unless the ETH fails. We defer detailed proof of this theorem to the Appendix A and only give a high-level idea here.

Theorem 2. DA-MULTICAST *is NP-hard even if the input graph is a planar graph of maximum degree six,* $|R_s| = b_s = 1$ *for each source s,* $c_a = 1$ *for each link a, and* $C = 0$.

We develop a reduction from VERTEX-DISJOINT PATHS on graphs with maximum degree three. To this end, let $I = (G = (V, E), (s_1, t_1), (s_2, t_2), \ldots, (s_k, t_k))$ be an instance. We start by replacing each edge $\{u, v\} \in E$ by two directed links (u, v) and (v, u). The capacity of each link is 1. We make each s_i and t_i both a sender and a receiver such that s_i is subscribed to t_i and t_i is subscribed to s_i. The bandwidth demand is always 1. Now, there is solution to the constructed instance of DA-MULTICAST with $C = 0$ if and only if there is a set of disjoint paths between all terminal pairs (s_i, t_i). We can also replace each terminal vertex by a small gadget such that no vertex is both a sender and a receiver. Next, we show that the problem remains hard, even if there is only a single receiver.

Proposition 2. DA-MULTICAST *is strongly NP-hard even if there is only a single receiver and* $C = 0$.

The full proof of the above proposition can be found in Appendix B. In summary, here, we reduce from BIN PACKING. For a given instance of BIN PACKING, we construct an equivalent instance of DA-MULTICAST as follows. We start with a source s_i for each item i and a single receiver r, where the bandwidth requirement b_{s_i} is the size of i and r is subscribed to each source. Next, we add a vertex v_j for each bin $j \in [k]$. We add a link of capacity B from each source s_i to each vertex v_j and from each vertex v_j to r. The idea is that the input instance is a yes-instance if and only if we can send b_{s_i} units from each source to some vertex v_j in such a way that each vertex v_j can send a combined of B units to r, that is, the input instance of BIN PACKING is a yes-instance.

C. Adding length bounds

In this section, we study a variant of DA-MULTICAST where the input contains an additional set L and we are only allowed to assign link-lengths in L. We call the problem LENGTH-BOUNDED NETWORK MULTICAST and note that DA-MULTICAST is the special case of it where all link-lengths are allowed. However, it is a priori not clear whether restricting L to some constant size makes the problem easier or harder. Due to space constraints, we include the detailed proof of this theorem in the full version of the paper and only give a high-level idea here.

Theorem 3. LENGTH-BOUNDED NETWORK MULTICAST *is NP-hard even if the input graph is planar and has maximum degree* 6, $|R_s| = b_s = 1$ *for each source s,* $R \cap S = \emptyset$, $c_a = 1$

Input Var.	Meaning
S	The set of sources
R^s	The set of receivers from source $s \in S$
B^s	Bandwidth requirement of a source
N^x	Neighbors of node x
$c_{(x,y)}$	capacity of the link (x, y)

MILP Var.	Meaning
C	Maximum congestion over all links
$w_{(x,y)}$	Weight of link (x, y)
$\mathrm{dist}_{x,y}$	Distance from node x to node y
$a_{(x,y)}$	Activity of link (x, y)
$a^s_{x\to y}$	Activity of directed link $x \to y$ from s
$a^{s,r}_{x\to y}$	Activity of directed link $x \to y$ from s to r
$a_{x,z,y}$	Activity of z from source s between x and y

Table I: A summary of variables used in the mixed integer linear program. First part of the table shows input variables, and the second part shows variables needed in the MILP.

for each link a, $C = 0$, and $L = \{1, 2\}$. Moreover assuming the ETH, this special case can not be solved in $2^{o(\sqrt{n+m})}$ time.

The idea for this hardness result is basically the same as for theorem 2 but some details have to handled in slightly different ways. In particular, we reduce directly from PLANAR 2P1N-3-SAT and combine the reduction from PLANAR 2P1N-3-SAT by Middendorf and Pfeiffer [27] with the reduction behind theorem 2 in a non-sequential way.

We mention in passing that if we do not require the input graph to be planar, then we can use the same reduction as in Theorem 3 but start from (non-planar) 2P1N-3-SAT to show that LENGTH-BOUNDED NETWORK MULTICAST (and also DA-MULTICAST) can not be solved in $2^{o(n+m)}$ time.

IV. EXACT SOLUTION

We now present an approach to compute exact solutions for our problem in super-polynomial time, using a Mixed Integer Linear Program (MILP). The optimal solutions obtained by our MILP (Program 1) can also serve as a baseline to compare our heuristics. A summary of parameters used in our MILP is presented in Table I. We start by briefly reviewing the variables of our MILP:

- **Total congestion.** Our goal is to minimize the amount of congestion in the network, which we denote by C.
- **Weight & distance variables.** To ensure that we achieve the shortest paths via the weight variable $w_{(x,y)}$ that we set for each link, we define an extra integer variable $\mathrm{dist}_{x,y}$ that determines the distance between any pairs of nodes $x, y \in V$.
- **Activity variables.** We define four *activity* indicator binary variables that determine if a link is active or not. Activity of a link means the (directed) link is being used as a part of a shortest path, either in general ($a_{(x,y)}$), from a source s ($a^s_{x\to y}$), or between a source s and a receiver r ($a^{s,r}_{x\to y}$).

Program 1: Mixed Integer Linear Program for DA-MULTICAST

```
1  Minimize $C$
2  $\mathrm{dist}_{x,x} = 0$                                    $\forall x \in V$
3  for $(x, y) \in E$    # Weight & distance constraints of a link
4      $a_{(x,y)} \in \{0, 1\}$
5      $w_{(x,y)} \geq 1$
6      $\mathrm{dist}_{x,y} \leq w_{(x,y)}$
7      $\mathrm{dist}_{x,y} \geq w_{(x,y)} - M \cdot (1 - a_{(x,y)})$
8  for $x, z, y \in V$    # Ensuring shortest distance between a pair of nodes
9      $\mathrm{dist}_{x,y} \leq \mathrm{dist}_{x,z} + \mathrm{dist}_{z,y}$
10     $a_{x,z,y} \in \{0, 1\}$
11     $\mathrm{dist}_{x,y} \geq \mathrm{dist}_{x,z} + \mathrm{dist}_{z,y} - M \cdot (1 - a_{x,z,y})$
12 $a_{(x,y)} + \sum_{z\in V} a_{x,z,y} \geq 1$                  $\forall x, y \in V$
13 for $\forall s \in S$    # Ensuring an active path from a source...
14     for $x \to y \in E$
15         $a^s_{x\to y} \in \{0, 1\}$
16         $a^s_{x\to y} \leq a_{(x,y)}$
17         $a^s_{y\to x} \leq a_{(x,y)}$
18     for $\forall r \in R^s$                         # ...to a receiver
19         for $x \to y \in E$
20             $a^{s,r}_{x\to y} \in \{0, 1\}$
21             $a^{s,r}_{x\to y} \leq a^s_{x\to y}$
22         $\sum_{x\in N(s)} a^{s,r}_{s\to x} = 1$
23         $\sum_{x\in N(y)\setminus s} a^{s,r}_{x\to y} = \sum_{x\in N(y)\setminus R^s} a^{s,r}_{y\to x}$   $\forall y \in V \setminus (\{s\} \cup R^s)$
24         $\sum_{x\in N(r)\setminus r} a^{s,r}_{x\to r} = 1$
25 $C \geq \sum_{s\in S}(a^s_{x\to y} \cdot B_s) - c_{(x,y)}$          $\forall (x, y) \in E$
26 $C \geq 0$
```

Our MILP aims to minimize congestion occurring in the given network (Line 1). We first ensure that the distance from all nodes to themselves is zero in Line 2.

To linearize the computation of the shortest path distance, the essential part is computing a minimum of two values. Let us consider v_1 and v_2 as our variables. We then consider an auxiliary variable $a \in \{0, 1\}$ and a large value M, bigger than any possible value that v_1 and v_2 can ever be assigned to[3]. We then can linearize the minimum of v_1 and v_2 as:

$$\min(v_1, v_2) \leq v_1, \ \min(v_1, v_2) \leq v_2$$

$$\min(v_1, v_2) \geq v_1 - M \cdot a, \ \min(v_1, v_2) \geq v_2 - M \cdot (1 - a)$$

We next discuss the main idea behind each block of code based on each of our three main for loops.

- **Link constraints.** We first define the binary activity variable for each link in Line 4. We then ensure that

[3] In practice, setting M larger than $2n$ times the maximum possible capacity is enough.

all of the links have the weight of at least 1 (Line 5). We also make sure that if a link has been selected as an active link, then the distance between its two endpoints should equal its weight (Lines 6 and 7).

- **Distance of a pair of nodes.** In this part of the code, we select the shortest path among all paths between nodes x and y, following the shortest path property

  $$\mathrm{dist}(x, y) = \min(\mathrm{dist}(x, y), \mathrm{dist}(x, z) + \mathrm{dist}(z, y))$$

  and applying the linearization of the minimum (see Lines 9 to 11).
- **Active path from a source to a receiver.** In this block of the program, we ensure that there is an active path from each source to each receiver. In particular, Lines 22 to 24 make sure that this path is unique and unsplittable from source s to receiver r. Lines 14 to 21 ensure that this path is part of the shortest path that has been constrained in the previous blocks of the program.

Furthermore, in Line 12, the underlying paths created by active parameters ensure reachability between any pairs of nodes. Finally, Line 25 sets the minimum congestion value as the least congestion on each link. Congestion on a link is based on the bandwidth of multicast trees that use this link minus the capacity of the link.

V. Heuristics

We next study how to design demand-aware multicast trees in large networks, using fast heuristics, and show their effects in an experimental evaluation.

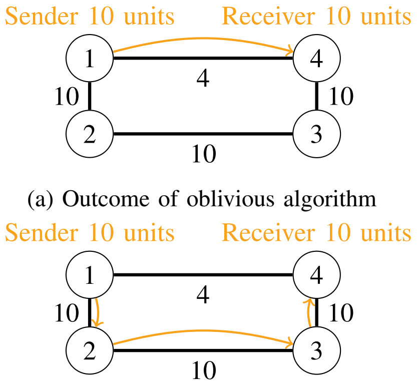
A. Algorithms

Before going into details of our algorithms, we first revisit the state-of-the-art demand-oblivious solution to set link weights and compare them to new, demand-aware algorithms.

Oblivious algorithm. The default practical approach to traffic engineering is to set link weights inversely proportional to the link capacities. To compare our algorithms with this strategy, we consider this approach as the first heuristic, by simply setting the weight of a link (x, y) to $w_{(x,y)} = \frac{1}{c_{(x,y)}}$. We know that the running time of this algorithm only depends on the number of edges, i.e., it is in $\Theta(m)$.

However, it is known that the outcome of this algorithm can be far from optimal. Figure 2 shows such an example with only two paths. This example can be extended, e.g. by adding more links on the bottom path and then reducing the capacity of the top link, to generate scenarios that the congestion can be arbitrarily large. A key observation in this and similar examples is that the oblivious algorithm sometimes sets small weights for links that it should not. This happens because the algorithm is *oblivious* to demand.

Therefore, we introduce two algorithms that are *demand-aware* by design. These two algorithms, DA-Picky and DA-Hybrid, rely on the idea of finding the largest possible bottleneck links: those links that probably result in congestion. We first detail the procedure of finding bottleneck links, before going into details of algorithms.



(a) Outcome of oblivious algorithm

(b) Outcome of our DA algorithms

Figure 2: The above figures show the resulting routes considering the oblivious and DA algorithms. As you can see in Figure 2a, on the link between nodes 1 and 4, 10 units of traffic pass while the link only has 4 units of capacity. This causes 6 units of congestion. Using our DA algorithms, we can make a route selection that prevents any congestion from occurring as shown in Figure 2b.

Algorithm 2: DA-Hybrid

```
Sort all sources by their demand in an descending
  manner.
for Source $s$
    Sort all links by their capacity.
    while We have less than $n - 1$ selected links do
        Select a link that does not create a cycle.
    Set weights of selected links in the to $\frac{1}{B_s}$.
    Reduce capacity of selected links by $B_S$.
Compute congestion and fall-back to oblivious
  algorithm if needed.
```

Bottleneck links. In a (multicast) tree, a bottleneck link is the link with the smallest capacity. To minimize congestion, an approach is to find a set of maximum-bottleneck spanning trees (MBSTs for short), and then overlapping them. To find a MBST we sort all links by their capacity and then construct the tree by adding links one by one until we have a spanning tree, i.e. $n - 1$ links without a cycle. Now, given this construction, we introduce our first algorithm, DA-Hybrid.

DA-Hybrid algorithm. This algorithm starts by sorting sources based on their demand. Let s be a source with bandwidth B_s. We consider that all weights are initially set to infinity.

We then iteratively find a MBST and set the weights of all links of the MBST (which have not been assigned weights before) to $\frac{1}{B_s}$. Furthermore, we reduce the capacity of all selected links by B_s. We then compute the congestion with the above weights, and fall back to the oblivious algorithm

Algorithm 3: DA-Picky

```
Sort all sources by their demand in an ascending manner.
for Source $s$
    while Not all of its receivers are reachable do
        Add the link with the smallest weight higher than the demand of $s$, with weight one divided by capacity.
    if Congestion of a link is bigger than $0$ then
        Add all the remaining links with weight one divided by capacity.
        Break.
```

if the resulting congestion is higher.

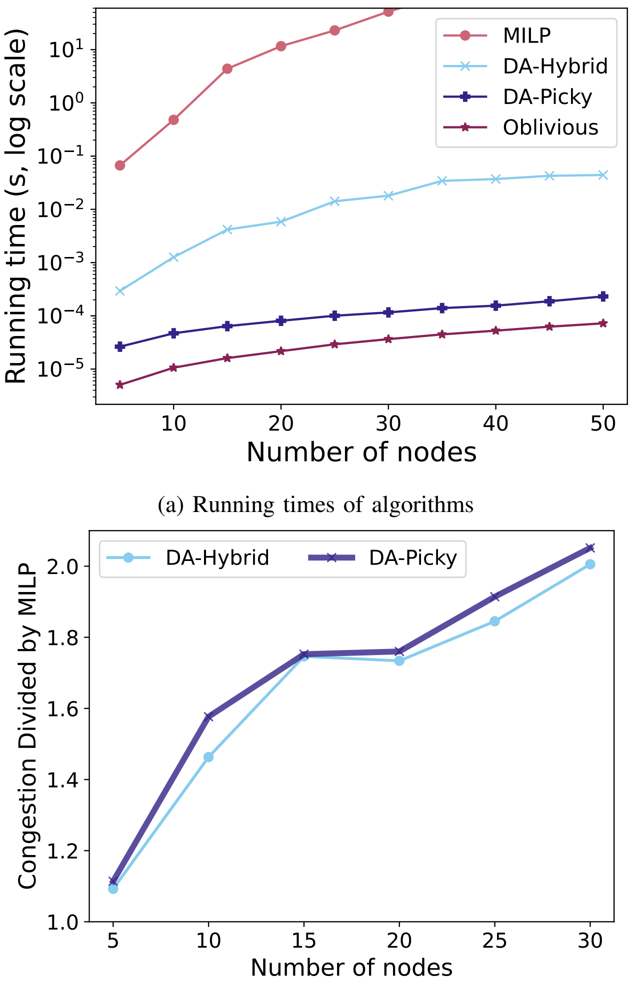
For this algorithm, we need to sort the links by capacity and nodes by their bandwidth once, which takes $\Theta(m \log m)$. Then computing MBST from each source takes $\Theta(m)$, and hence for all sources it takes $\Theta(m|S|)$, where $|S|$ indicates the number of sources. Also, the fall-back mechanism requires a running time of at most $\theta(m)$ Thus, the total running time of this algorithm is $\Theta(m \cdot \max(\log m, |S|))$.

DA-Picky. In this algorithm, we similarly start by considering all weights to be infinity and also start at the source with the highest demand. Furthermore, in each iteration, we first check if previously set weights are enough to route to receivers of a source or not (i.e. if there are paths with non-infinity values). If previous weights were not enough, we add links with the highest capacity higher than the demand of the source, assigning weight one divided by their capacity, until all receivers become reachable. At this point, we compute the congestion on all links, and if congestion on any of the links is bigger than 0, we add all the remaining links with weight one divided by capacity. Algorithm 3 describes a pseudocode of this algorithm.

For the DA-Picky algorithm we need to sort the demand of sources and capacity of links once, which requires a running time of $\Theta(m \cdot \log m)$. Then in each iteration, we need to add links one by one and check reachability each time, which requires a running time of $\Theta(m \cdot |S|)$. Hence, the total running time of this algorithm, similar to DA-Hybrid is $\Theta(m \cdot \max(\log m, |S|))$.

By design, both DA-Hybrid and DA-Picky always ensure congestion less than equal to the oblivious algorithm. This is true because when it faces congestion bigger than equal to zero, it falls back to the oblivious algorithm.

Observation 4. *Using weights assigned by DA-Picky, congestion in the system is always less than equal to the oblivious algorithm.*



(a) Running times of algorithms

(b) Approximation value of DA algorithms

Figure 3: Figure 3a shows the running time of our MILP and three heuristics, with increasing size of the graph generated by the IGen networks [28]. Running times up to a minute are shown using a log-ratio. Figure 3b shows the approximation ratio of our demand-aware algorithms, on instances generated by IGen.

VI. Experimental Evaluations

In this section we detail our experimental results[4]. In particular, we discuss and answer the following questions:

Q1. How long does it take to run our algorithms?

Q2. How well do our algorithms perform compared to MILP?

Q3. Which algorithm performs better on real-world networks?

To answer these questions, we first discuss the network topologies that we used and the approaches we took to set relevant parameters. In the end, we discuss the results and answers to the thee mentioned questions.

[4] We have open-sourced our code which is accessible via our GitHub repository.

Network topologies. In our experiments, we have used various real-world and synthetic network topologies in our evaluations. Part of our results are based on networks obtained from The Internet Topology Zoo data set [29] (we call it *zoo topologies* from now on). This dataset of networks reflects real-world topologies. However, the included topologies are quite small in size. To have a larger variety of network topologies, we created additional topologies that emulate real-world scenarios based on the IGen topology generation tool [28]. Furthermore, to represent smaller networks, we manually created a mesh topology that is typical for single-site campus networks [30]. Finally, we use a proprietary data set of a large European ISP, which we call the "national ISP" dataset. The set of sources and receivers for each sender were selected randomly.

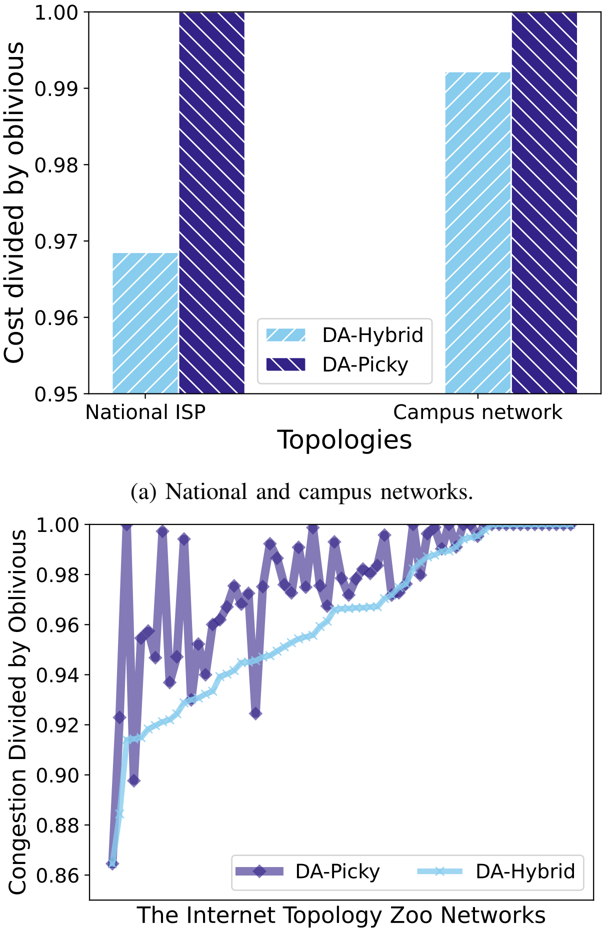
Source bandwidth and link capacities To assign the bandwidth demand of each source and the capacity of each link, we have performed the following steps for the respective topologies.

- **Zoo topologies.** The bandwidth demands and link capacities were sampled uniformly at random in the range from 10 to 100. In our experiments, the relation between capacity and bandwidth depends on the number of senders, and it can be adjusted by a ratio parameter.
- **IGen.** The IGen tool allows us to generate link capacities as well. It differentiates between core and access links, giving different bandwidths to each link type. We then sampled the bandwidth of each source uniformly at random similar to the zoo topologies.
- **National ISP network.** This dataset contains a capacity value for each link. To generate the bandwidth required for each source, we used the gravity model suggested by Roughan [31]. We were able to use this model as the population for each node in this data set was known. Concretely, we set the required demand for two cities with populations p_1 and p_2 as $\frac{p_1 \cdot p_2}{f}$, where f is a "friction parameter". As the gravity model gives bandwidth for pairs of nodes, we then normalize the bandwidth for each source by taking an average of its bandwidth to all its receivers.

We point out that our scenarios considered a static bandwidth for all sources. We believe that in case of dynamic changes in the demand, our core algorithm can be adjusted to respond fast to those changes by ideas suggested by prior works on dynamic spanning tree computation [32].

A. Results

In this section, we go over the results of our evaluations, focusing on answering the questions proposed earlier. Our programs are written in python 3.10, benefiting from networkx [33], gurobipy [34], and Matplotlib [35] libraries. They were executed on a machine with Intel® Xeon® CPU E5-1620 CPU with a clock frequency of 3.60GHz, and 64GB RAM. In our evaluations, given the inherent randomness of input generations, we repeat all instances at least 10 times.



(a) National and campus networks.

(b) Performance compared to oblivious

Figure 4: Figure 4a Shows the average cost over multiple runs on the national ISP data with real-world capacities and bandwidth requirements created using the gravity model based on population and a campus network with unit link capacities and bandwidth requirements sampled uniformly at random. Figure 4b shows the (multiplicative) approximation factor of the algorithms compared to the optimal solutions computed by the MILP for various topologies from the Topology Zoo [29] that are small enough to run our MILP on. Cases where the algorithms computed an optimal solution are omitted.

A1. Running time. Given the wide flexibility of the IGen dataset, here we report the running time of our algorithms on graphs with an increasing number of nodes, in a log-plot. As expected, the running time of MILP grows exponentially and is significantly larger than the rest. In our experiments, the DA-Picky has a running time similar to the oblivious approach. We believe this could be due to the fast fall-back mechanism that this algorithm has, compared to DA-Hybrid. Both heuristics provide good performance in practice, as shown in Figure 3a, and as we show later, we saw a trade-off between the running time and performance of our two demand-aware algorithms.

A2. Approximation ratio. For this experiment, we also relied on IGen topologies. As can be seen in Figure 3b, DA-Hybrid performs slightly better than DA-Picky algorithm. On the other hand, the approximation ratio of both algorithms grows. This behavior is expected, since the MILP is able to explore a wider range of possible weight assignments, and by our theoretical results that showed finding an approximation algorithm is not possible, even in restricted cases.

A3. Real-world networks. We observe that our DA-Hybrid algorithm provides better average congestion over multiple runs on the national ISP data set as can be seen in Figure 4a. Considering a large variant of Zoo topologies, we see in Figure 4b that DA-Hybrid produces solutions that in most cases out-perform the DA-Picky. We report that we have tested this setup for a wide range of bandwidth-capacity ratios, and in our tests, we saw similar results to what is depicted in the figure.

VII. Conclusion

This paper was focused on IP multicast, and the possibilities and challenges of incorporating the senders' bandwidth requirement into route selection algorithms of an IP multicast instance. Our first result indicates that finding the optimal link weights for a multicast instance is NP-hard, even for cases that are restricted to binary weights. We then proposed efficient solutions based on a connection to maximum-bottleneck spanning trees (MBST). We further conducted experiments on a wide variety of inputs, showing benefits of our demand-aware algorithms compared to the traditional demand-oblivious algorithm.

Overall, we have shown that taking the easily accessible and thus far unutilized information of sender bandwidth demands into account when constructing multicast trees can help to reduce the overall congestion occurring inside a network. As a future work we aim to go beyond static topologies, giving flexibility for senders or receivers to join dynamically, in an online manner. In such a case, reconstruction of all trees should be done more adaptively. We also plan to further explore approximation algorithms for our problem using randomized approaches.

References

[1] M. Hu, J. Li, C. Cai, T. Deng, W. Xu, and Y. Dong, "Software defined multicast for large-scale multi-layer LEO satellite networks," *IEEE Trans. Netw. Serv. Manag.*, 2022.

[2] M. Zhang, "The design of multimedia multicast system in campus network based on three-tier architecture," in *ACM ICMSS*, 2021.

[3] K. P. Birman, "A review of experiences with reliable multicast," *Softw. Pract. Exper.*, 1999.

[4] L. Giuliano, C. Lenart, and R. Adam, "TreeDN: Tree-Based Content Delivery Network (CDN) for Live Streaming to Mass Audiences," RFC 9706, Jan. 2025. [Online]. Available: https://www.rfc-editor.org/info/rfc9706

[5] I. Wijnands, E. C. Rosen, A. Dolganow, T. Przygienda, and S. Aldrin, "Multicast using bit index explicit replication (BIER)," *RFC*, 2017.

[6] B. Fenner, M. J. Handley, H. Holbrook, I. Kouvelas, R. Parekh, Z. Zhang, and L. Zheng, "Protocol independent multicast - sparse mode (PIM-SM): protocol specification (revised)," *RFC*, 2016.

[7] J. D. Boeck, B. Fortz, and S. Schmid, "The case for stochastic online segment routing under demand uncertainty," in *IFIP Networking*, 2023.

[8] M. Parham, T. Fenz, N. Süss, K. Foerster, and S. Schmid, "Traffic engineering with joint link weight and segment optimization," in *ACM CoNEXT*, 2021.

[9] A. Brundiers, T. Schüller, and N. Aschenbruck, "Tactical traffic engineering with segment routing midpoint optimization," in *IFIP Networking*, 2023.

[10] ——, "Preprocess your paths-speeding up linear programming-based optimization for segment routing traffic engineering," in *IFIP Networking*, 2024.

[11] M. Henzinger, A. Paz, A. Pourdamghani, and S. Schmid, "The augmentation-speed tradeoff for consistent network updates," in *ACM SOSR*, 2022.

[12] S. Agarwal, M. S. Kodialam, and T. V. Lakshman, "Traffic engineering in software defined networks," in *IEEE INFOCOM*, 2013.

[13] N. Feamster, J. Rexford, and E. W. Zegura, "The road to SDN: an intellectual history of programmable networks," *Comput. Commun. Rev.*, 2014.

[14] M. Chiesa, G. Kindler, and M. Schapira, "Traffic engineering with equal-cost-multipath: An algorithmic perspective," *IEEE/ACM Trans. Netw.*, 2017.

[15] M. Rost and S. Schmid, "Virtucast: Multicast and aggregation with in-network processing: An exact single-commodity algorithm," in *Principles of Distributed Systems: 17th International Conference, OPODIS 2013, Nice, France, December 16-18, 2013. Proceedings 17*. Springer, 2013, pp. 221–235.

[16] B. Fortz and M. Thorup, "Increasing internet capacity using local search," *Comput. Optim. Appl.*, 2004.

[17] R. Ramanathan, "Multicast tree generation in networks with asymmetric links," *IEEE/ACM Trans. Netw.*, 1996.

[18] M. S. Kodialam, T. V. Lakshman, and S. Sengupta, "Online multicast routing with bandwidth guarantees: a new approach using multicast network flow," *IEEE/ACM Trans. Netw.*, 2003.

[19] C. P. Low and N. Wang, "An efficient algorithm for group multicast routing with bandwidth reservation," *Comput. Commun.*, vol. 23, no. 18, pp. 1740–1746, 2000.

[20] N. Wang and G. Pavlou, "Traffic engineered multicast content delivery without MPLS overlay," *IEEE Trans. Multim.*, 2007.

[21] M. Franke, J. Holland, and S. Schmid, "MCQUIC - A multicast extension for QUIC," in *NCA*, 2024.

[22] L. Georgiadis, "Bottleneck multicast trees in linear time," *IEEE Commun. Lett.*, 2003.

[23] R. Cohen and G. Kaempfer, "A unicast-based approach for streaming multicast," in *IEEE INFOCOM*. IEEE Comptuer Society, 2001.

[24] A. Young, J. Chen, Z. Ma, A. Krishnamurthy, L. L. Peterson, and R. Wang, "Overlay mesh construction using interleaved spanning trees," in *IEEE INFOCOM*, 2004.

[25] M. Kucharzak and K. Walkowiak, "Modeling and optimization of maximum flow survivable overlay multicast with predefined routing trees," *Telecommun. Syst.*, 2014.

[26] G. Rodolakis, A. Laouiti, P. Jacquet, and A. M. Naimi, "Multicast overlay spanning trees in ad hoc networks: Capacity bounds, protocol design and performance evaluation," *Comput. Commun.*, 2008.

[27] M. Middendorf and F. Pfeiffer, "On the complexity of the disjoint paths problems," *Combinatorica*, 1993.

[28] B. Quoitin, V. V. den Schrieck, P. François, and O. Bonaventure, "Igen: Generation of router-level internet topologies through network design heuristics," in *IEEE ITC*, 2009.

[29] S. Knight, H. X. Nguyen, N. Falkner, R. A. Bowden, and M. Roughan, "The internet topology zoo," *IEEE JSAC*, 2011.

[30] M. N. B. Ali, M. E. Hossain, and M. M. Parvez, "Design and implementation of a secure campus network," *IJETAE*, 2015.

[31] M. Roughan, "Simplifying the synthesis of internet traffic matrices," *Comput. Commun. Rev.*, 2005.

[32] C. Cheng, I. A. Cimet, and S. P. R. Kumar, "A protocol to maintain a minimum spanning tree in a dynamic topology," in *ACM SIGCOMM*, 1988.

[33] A. A. Hagberg, D. A. Schult, and P. J. Swart, "Exploring network structure, dynamics, and function using networkx," in *Proceedings of the 7th Python in Science Conference*, 2008.

[34] Gurobi Optimization, LLC, "Gurobi Optimizer Reference Manual," 2023. [Online]. Available: https://www.gurobi.com

[35] J. D. Hunter, "Matplotlib: A 2d graphics environment," *Comput. Sci. Eng.*, 2007.

[36] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman, 1979.

Appendix

A. Proof of Theorem 2

Proof. We present a reduction from Vertex-Disjoint Paths on graphs with maximum degree three where all terminal vertices have degree two.

Hence, let $I = (G = (V, E), (s_1, t_1), (s_2, t_2), \dots, (s_k, t_k))$ be an instance and let $T = \{s_1, s_2, \dots, s_k, t_1, t_2, \dots, t_k\}$ be the set of terminal vertices. We construct an equivalent instance of DA-Multicast as follows. We start by replacing each edge $\{u, v\} \in E$ by two directed arcs (u, v) and (v, u). We then set $c_a = 1$ for each arc a and $C = 0$. Next, we set $S = T$, $R = T$, $R_{s_i} = \{t_i\}$, $R_{t_i} = \{s_i\}$ and $b_s = 1$ for each vertex $s \in T$. This concludes the construction. Since the reduction can clearly be computed in polynomial time, it remains to show the correctness. To this end, assume that I is a yes-instance of Vertex-Disjoint Paths. Then, there exist a set of k vertex-disjoint paths, one path P_i between each pair (s_i, t_i). Setting the weight of all arcs (u, v) and (v, u) for which $\{u, v\}$ appears in a path P_i to 1 and the weight of all other arcs to n, the unique shortest path from s_i to t_i has length at most $n - 1$. Hence, each source s_i will only send data via the path P_i and each source t_i will only use the reverse edges of P_i. Since no two paths P_i and P_j share an arc, it holds for each arc a that

$$\sum_{s \in S} (b_s \cdot \max_{r \in R_s} \mathbf{1}_{a \in P_{s,r}}) \le \sum_{s \in S} \mathbf{1}_{a \in P_{s,r}} \le 1 \le c_a + C.$$

That is, the constructed instance of DA-Multicast is a yes-instance.

For the other direction, assume that the constructed instance of our problem, DA-Multicast, is a yes-instance. Then, we can assign weights to all arcs such that for all $i \neq j$, there is a unique shortest paths Q_i from s_i to t_i and a unique shortest path Q_i' from t_i to s_i. Note that Q_i is the reverse of Q_i' since the shortest path is unique and each path has the same length as its reverse since $w_{(u,v)} = w_{(v,u)}$ for all arcs (u, v). Moreover for each pair $i \neq j$, it holds that Q_i does not share an arc with Q_j or Q_j' since such an intersection would lead to congestion at least one. Hence, the path P_i between s_i and t_i corresponding to Q_i in G is edge-disjoint from the path P_j between s_j and t_j in G corresponding to Q_j. Note that in (undirected) graphs of maximum degree three it holds for any four vertices s_1, s_2, t_1, t_2 that if these vertices all have degree at most two, then any s_1-t_1-path P is edge-disjoint from any s_2-t_2-path Q if and only if P and Q are vertex-disjoint. Thus, P_i and P_j are vertex-disjoint. Since this holds for all pairs $i \neq j$, the instance I of Planar Vertex-Disjoint Paths is a yes-instance. This concludes the proof. □

B. Proof of Proposition 2.

Proof. We reduce from Bin Packing. Here, we are given a multiset X of integers and two integers k and B. The question is whether X can be partitioned into k multisets such that the sum of integers in each part of the partition is at most B. It is known that Bin Packing remains NP-hard even if the sum of integers in X equals $k \cdot B$ and all integers are encoded in unary [36]. For a given instance $I = (X = \{x_1, x_2, \dots, x_n\}, k, B)$ of Bin Packing with the mentioned restrictions, we construct an equivalent instance of DA-Multicast as follows. We start with a source s_i for each element x_i of X and a single receiver r. Next, we set $b_{s_i} = x_i$ and $R_{s_i} = \{r\}$ for each source s_i and $C = 0$. We continue by constructing G by adding vertices v_j for each $j \in [k]$. The set of arcs in G is $\{(s_i, v_j), (v_j, s_i), (v_j, r), (r, v_j) \mid i \in [n] \wedge j \in [k]\}$. The capacities of all arcs is B.

Since the instance can clearly be computed in polynomial time, it remains to show that the two instances are equivalent. To this end, first assume that I is a yes-instance. Then, there exists a partition of X into k parts $Y_1, Y_2, \dots, Y_k$ each summing to exactly B. Setting the lengths of all arcs incident to r to one and the length of arcs $(s_i, v_j), (v_j, s_i)$ to one if x_i belongs to the j^{th} part and to 2 otherwise leads to a solution with $C = 0$ since each arc incident to a source s_i has capacity $B \ge x_i = b_{s_i}$ and each arc (v_j, r) also has capacity $B = \sum_{x_i \in Y_j} x_i = \sum_{x_i \in Y_j} b_{s_i}$.

For the other direction, assume that the constructed instance of DA-Multicast is a yes-instance. Note that $\sum_{s_i} b_{s_i} = k \cdot B$ and r has exactly k incoming arcs each with capacity B. Hence, each of these arcs has to be fully saturated in any solution with $C = 0$. This implies that there is a way to partition the sources into k parts $Z_1, Z_2, \dots, Z_k$ such that for each $j \in [k]$ it holds that $\sum_{s_i \in Z_j} b_{s_i} = B$. This corresponds to a solution for I. □