Computer Communications 164 (2020) 138–147

Contents lists available at ScienceDirect

Computer Communications

journal homepage: www.elsevier.com/locate/comcom

Efficient non-segregated routing for reconfigurable demand-aware networks

Thomas Fenz [a], Klaus-Tycho Foerster [a,*], Stefan Schmid [a], Anaïs Villedieu [b]

[a] *Faculty of Computer Science, University of Vienna, Austria*
[b] *Institute of Logic and Computation, Technische Universität Wien, Austria*

ARTICLE INFO

Keywords:
Reconfigurable networks
Algorithms
Routing
Programmable physical layer
Evaluations

ABSTRACT

More and more networks are becoming *reconfigurable*: not just the routing can be programmed, but the physical layer itself as well. Various technologies enable this programmability, ranging from optical circuit switches to beamformed wireless connections and free-space optical interconnects. Existing reconfigurable network topologies are typically *hybrid* in nature, consisting of static and a reconfigurable links. However, even though the static and reconfigurable links form a joint structure, routing policies are artificially segregated and hence do not fully exploit the network resources: the state of the art is to route large elephant flows on direct reconfigurable links, whereas the remaining traffic is left to the static network topology. Recent work showed that such artificial segregation is inefficient, but did not provide the tools to actually leverage the benefits on non-segregated routing.

In this paper, we provide several algorithms which take advantage of non-segregated routing, by *jointly* optimizing topology and routing. We compare our algorithms to segregated routing policies and also evaluate their performance in workload-driven simulations, based on real-world traffic traces. We find that our algorithms do not only outperform segregated routing policies, in various settings, but also come close to the optimal solution, computed by a integer linear program formulation, also presented in this paper. Finally, we also provide insights into the complexity of the underlying combinatorial optimization problem, by deriving approximation hardness results.

1. Introduction

The fast growth of machine learning and artificial intelligence applications will soon lead to a significant increase of data-intensive workloads, and hence more traffic in datacenters [1]. The latter hence become a critical infrastructure of our digital society.

While the design of cost-effective datacenter networks providing a high connectivity has received much attention over the last years already (e.g., [2–7]), we recently witness a trend to enhance traditional *static* datacenter networks with *reconfigurable* links: technological advances in reconfigurable optical switches and free-space optics allow to adjust datacenters (e.g., [8,9]) to the workload they serve, making them "demand-aware" [10].

While reconfigurable datacenter networks can be used to adjust to, and hence exploit, the typically *sparse and skewed* [9,11–13] nature of datacenter traffic, and hence provide shorter paths between frequent communication partners, today, we lack good algorithms for designing and routing on such networks. In particular, most existing literature only considers restricted "segregated" routing models where traffic is forced to either use the fixed network or a *single* reconfigurable link (see e.g., [9,14]), or their algorithms rely on overly simple matching-cased heuristics (see, e.g., [8,15–18]). While it is known that non-segregated routing improves the performance over purely segregated routing policies [14], the landscape of corresponding algorithms is hence mostly uncharted. In this paper we take the first steps to provide and evaluate new algorithms that benefit from the paradigm of non-segregation, reaping the perks of utilizing both hybrid network parts as a joint resource (see Fig. 1).

1.1. Our contributions

This paper presents algorithms for jointly optimizing topology and (non-segregated) routing, to build *demand-aware* networks which fully exploit the available resources and flexibilities. Our study encompasses complexity, algorithms, and work-load-driven simulation for such emerging networks:

1. *Complexity:* We prove that *even the approximation* of demand-aware network designs with non-segregated routing is NP-hard, by providing logarithmic inapproximability bounds. Additionally, we show that the scenarios of one and multiple reconfigurable switches are polynomial-time equivalent.

[*] Corresponding author.
E-mail addresses: thomas.fenz@univie.ac.at (T. Fenz), klaus-tycho.foerster@univie.ac.at (K.-T. Foerster), stefan_schmid@univie.ac.at (S. Schmid), avilledieu@ac.tuwien.ac.at (A. Villedieu).

https://doi.org/10.1016/j.comcom.2020.10.003
Received 4 February 2020; Received in revised form 3 September 2020; Accepted 7 October 2020
Available online 10 October 2020

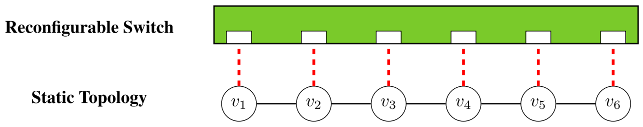


Fig. 1. In this small example, six nodes $v_1, \dots, v_6$ are connected to a reconfigurable switch (connections dashed): the network operator can choose a matching of nodes inside the switch, creating a direct path between two nodes each time. Consider the case where there is a demand $v_1 \rightarrow v_5$ and a demand $v_6 \rightarrow v_1$. Under *segregated* routing policies, one of the two demands (*e.g.,* $v_6 \rightarrow v_1$) can be routed directly via the reconfigurable switch, whereas the other demand must be routed inefficiently via the static topology. However, using *non-segregated* routing, the demand *e.g.,* $v_1 \rightarrow v_5$ may also use the direct matching connection from v_1 to v_6 as well for better efficiency, requiring just one more hop in the static topology on the link (v_6, v_5).

2. *Algorithms:* Given the hardness results, we present several polynomial-time heuristic algorithms as well as an exact algorithm based on a integer linear program (ILP) formulation.
3. *Empirical results:* Using multiple workload-driven simulations (based on Facebook, Microsoft, high performance computing, and pFabric traces), we compare our algorithms to state-of-the-art networks based on segregated routing schemes. Our algorithms significantly outperform segregated routing methods, coming close to optimal ILP solutions.

1.2. Organization

We describe our formal model in Section 2 and provide inapproximability NP-hardness results in Section 3. We then discuss various routing algorithms in Section 4, ranging from a integer linear program to greedy heuristics. The performance of these algorithms is then evaluated in Section 5, where we use segregated routing as a baseline. Lastly, we discuss related works in Section 6 and conclude in Section 7.

2. Model

We study the problem of computing a topology to optimally serve a given communication pattern, where the topology combines static (fixed) and reconfigurable links and can be *jointly* optimized together by non-segregated routing. Our model closely follows the notation and definitions of [14].

Network model. Let $N = (V, E, S, w)$ be a weighted *hybrid* network [18,19] connecting the n nodes $V = \{v_1, \dots, v_n\}$ (*e.g.,* top-of-the-rack switches), using (1) (usually electrically packet-switched) static links E and (2) optical links implemented through an reconfigurable optical circuit switch S.

A reconfigurable switch S connects a set of nodes $V' \subseteq V$ by choosing a matching M on V', where two matched nodes are connected by a bidirectional (undirected) link. We will also consider the directed case, where each node may have one incoming and one outgoing matching link. For the sake of generality, we assume each link, whether electrical or optical, comes with a non-negative weight w (a cost, *e.g.,* latency).

Generality. Our results also apply to non-optical switches and links, as long as they match the theoretical properties described in the model. As such, we will only talk about reconfigurable switches and links, simply implying any appropriate technology that matches our model. Moreover, as we discuss in Section 3, under non-segregated routing in weighted hybrid networks (the model we consider), the cases of one or multiple reconfigurable switches can be easily translated to another. We thus choose the case of one switch for ease of presentation.

Traffic demands. The resulting network should serve a certain communication pattern, represented as a $|V| \times |V|$ communication matrix D (the *demand matrix*) with positive real-valued entries. An entry (i, j) in D represents the traffic load (frequency) or demands from the node v_i to the node v_j.

Reconfiguration problem. We say that the hybrid network N is *configured* by the reconfigurable switches. That is, we will refer to the set of configured links $\mathcal{M} = \cup_{\ell=1}^{n} M_\ell$, the union of the matchings provided by the reconfigurable switches, as the *configuration* of N. For ease of notation, we will simply write $N(\mathcal{M})$ to denote the concrete topology resulting from configuration $\mathcal{M}$ and define $dist_{N(\mathcal{M})}(i, j)$ to be the shortest (weighted) distance from node v_i to node v_j on the network $N(\mathcal{M})$. Given a hybrid network N and communication demands D, our goal is to compute a network $N(\mathcal{M})$ which minimizes the (weighted) average path length for serving D in N by providing a set of matchings $\mathcal{M}$ accordingly. Succinctly stated:

$$\min \sum_{(i,j)\in D} D[i,j] \cdot dist_{N(\mathcal{M})}(i,j) \tag{1}$$

That is, we want to minimize the sum of the path lengths, weighted by the demand (*i.e.,* flow size) and link costs: for each ordered pair of nodes $v_i, v_j \in V$, we multiply the (weighted) length of the shortest path $dist_{N(\mathcal{M})}(i, j)$ from v_i to v_j on $N(\mathcal{M})$ with their entry (i, j) in D.

3. Hardness results

Before discussing NP-hardness results in this section, we first show that the cases of one or many reconfigurable switches are polynomially equivalent, by using link weights.

Problem transformation: many to one. At first sight, the case of multiple switches seems fundamentally different to just one reconfigurable switch: some nodes might be connected to multiple switches, which in turn might be connected to different subsets, creating complicated combinatorial dependencies.

However, we can translate these dependencies in a few steps. For each node v connected to k reconfigurable switches, we create k nodes $v_1, \dots, v_k$, connecting them to v with static links of weight 0. A newly created node v_i will be used to represent the connection of v to its ith reconfigurable switch S_i. We re-create the possible reconfigurable links from S_i as follows, slightly abusing notation: for all w connected to S_i, if v was able to connect to w via S_i with a weight of c_i, then v_i will be able to connect to w_i with a cost of c_i as well. However, all possible connections from v_i to other nodes, which were not originally possible from v via S_i, get assigned a prohibitively large weight. In turn, all original nodes are either disconnected from the new single reconfigurable switch S or receive the same large weights for all their possible reconfigurable links.

Hence, we can easily translate solutions obtained on this modified instance back to the case of multiple switches. Respectively, if even one reconfigurable link of prohibitively large weight is chosen for routing, we can conclude that the original instance was infeasible. We note that this transformation also allows us to directly transfer NP-hardness results from multiple switches to the case of one reconfigurable switch, for, *e.g.,* [14, Theorem 4], which showed NP-hardness for 2 demands from a min–max perspective.

Prior work [14, Theorem 3] already showed demand aware-routing in weighted hybrid networks to be NP-hard for a single reconfigurable switch, but did not provide approximation bounds. We now show that the objective value (1) of the optimal solution cannot be approximated better than $\Omega(\log n)$ for n nodes. Our reduction will be from `Dominating Set`, which has a logarithmic approximation bound [20].

Theorem 1. *Demand-aware routing cannot be approximated better than $\Omega(\log n)$, unless $P = NP$.*

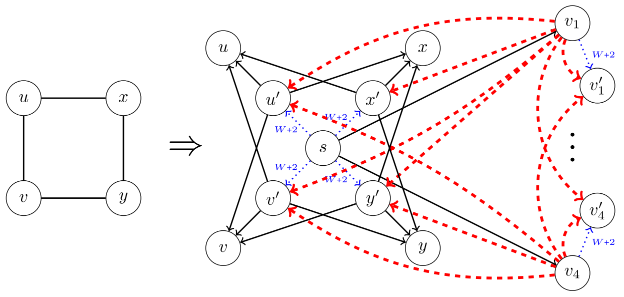


Fig. 2. In this small example, the four node graph on the left (instance I) is transformed to the $4 \cdot 4 + 1$ node graph on the right (instance I'). In I', all unmarked links (static or reconfigurable ones, dashed) have a cost of 1; reconfigurable links with a weight of greater than $W+2$ are omitted, and static links with a weight of $W+2$ are dotted. We also omit the reconfigurable switches for less clutter. The node s has demands of 1 to u, v, x, y and $v_1', \dots, v_4'$, which each have a cost of $W+3$ if only static links are used. In order to improve the routing, the possible reconfigurable links of cost 1 can be used to build shortcuts for the demands, reaching a cost of $1+1+1=3$ each time. Note that each node from $v_1, \dots, v_4$ can only create one outgoing reconfigurable link. When matching to nodes from $v_1', \dots, v_4'$, only one such shortcut is created, but when a match to a node v' is made, then all nodes which v dominates in I obtain a shortcut. Hence, the optimal solution corresponds to an optimal dominating set in I, where each extra node needed to dominate induces a penalty of W: here I can be dominated by two nodes, *e.g.,* x and v, by matching to x' and v' in I', only two demands to $v_1', \dots, v_4'$ need to have an expensive route, which corresponds to the dominating set size in I. As such, if the objective function penalty in I' has less than logarithmic overhead, compared to the optimal solution, we can approximate the domination process in I better than logarithmically as well.

Proof. Feige showed that `Dominating Set` cannot be approximated better than $\Omega(\log n)$, unless $P = NP$ [20]. That is, given a graph $G = (V, E)$, find a set $K \subseteq V$ of minimum cardinality k s.t. every $v \in V$ is in K or has a neighbor in K.

Let I be an instance $G = (V, E)$ of `Dominating Set`. We construct an instance I' ($G' = (V', E')$) of a modified demand-aware topology design and routing problem, where a node may be connected to multiple reconfigurable switches, which in turn just connect a subset of the nodes. Recall that we showed this problem to be polynomial-time equivalent.

We begin with the static network in G'. We first duplicate all nodes $v \in V$ and denote their clone by v', creating directed links of cost 1 from v' to v. Next, if (u, v) is a link in E, then we create a directed link with cost 1 from u' to v, iterating this over all links in E and adding them to E'. However, E' will not contain the links from E. Moreover, we create the nodes s, $v_1, v_2, \dots, v_{|V|}$, and $v_1', v_2', \dots, v_{|V|}'$, connecting s to each of the nodes $v_1, v_2, \dots, v_{|V|}$ with a directed link of cost 1, and each v_i to its respective v_i' with a directed link of cost $W + 2 > 2$, to be specified later. Lastly, s is connected to each cloned node v' with a directed link of cost $W + 2$. Before specifying the reconfigurable part, we create the demands D: s has a demand of 1 to each node v_i' and to each node v originating from V and no further demands exist. In the static network alone, the routing cost to each of these $2|V|$ nodes is $1 + W + 2$, *i.e.,* $2|V|(1 + W + 2)$ in total.

For the reconfigurable switches, we create $|V|$ of those, each connecting a node v_i with all $|V|$ cloned nodes v' and all $|V|$ nodes v_i. However, only the outgoing links of v' have a cost of 1, all other outgoing possible reconfigurable links have a cost of $> W + 2$. As such, if any reconfigurable link is used for routing a positive demand that is not outgoing from a node v_i, then this route has a cost of at least $1 + W + 2$, *i.e.,* not cheaper than solely using the static network. An example of the polynomial construction process is given in Fig. 2.

Hence, the only remaining decisions are where to match the outgoing links of the nodes v_i, as they allow routes of cost 3: by matching to nodes of type v_i', one demand has a cost of 3, whereas by matching to a node v', all nodes that v would dominate in G (which is at least v itself), can be routed to with a cost of 3. In order to obtain an optimal solution for I', the task is to cover the nodes v' with as few matching links as possible. If K' such links, with $|K'| = k'$, suffice, s.t. all nodes v (originating in V) can be reached from s with a cost of 3, then only k' nodes from v_i' need to have their demand routed with a cost of $1+W+2$. In other words, an optimal solution minimizes k', with a routing cost of $k'(3 + W) + 3(2|V| - k') = k'W + 6|V|$. Optimal k' for I' and k for I must have the same size, as k' matching links (covering all nodes v originating in V) represent a dominating set for G and vice versa.

It remains to show the transfer of the inapproximability results, which we prove in the spirit of a linear reduction [21]. To this end, we pick W sufficiently large, *e.g.,* $W = 100|V|^2$, one can easily optimize for significantly smaller values of W while retaining the same results. Assume that our optimization problem can be approximated better than $\Omega(\log n)$, by some approximation ratio f. We can then approximate `Dominating Set` better than $\Omega(\log n)$ as well by (1) picking the `Dominating Set` instance I, (2) creating the corresponding instance I', (3) approximating I' with ratio f, and then (4) positively answering that a solution of size fk exists for I, by observing the following: A solution of size at most $fk'W + f6|V|$ of I' implies that there is a solution for I' where at most $\lfloor fk' \rfloor$ demands have to be routed with a cost of at least $1 + W + 2$, as W significantly exceeds $f6|V|$. Hence, we conclude that there is also a solution of I using at most $\lfloor fk' \rfloor$ nodes to dominate the graph, which would imply $P = NP$, as f is not contained in $\Omega(\log n)$. □

4. Algorithms

Given that computing an optimal non-segregated routing on a reconfigurable network is NP-hard to approximate, we next present various polynomial-time heuristics as well as a non-polynomial time *exact* algorithm. We start with a general ILP in Section 4.1, followed by an algorithm for a single flow in Section 4.2. We then provide a wide range of polynomial-time general heuristics, starting by prioritizing large demands in Section 4.3 respectively large demands w.r.t. their initial routing distance in Section 4.4, followed by algorithms that greedily add paths 4.5 or links (Section 4.6), concluding in Section 4.7.

4.1. An ILP for demand-aware routing

The fundamental idea is that we would like to select a matching in the reconfigurable switch that optimizes the objective function for the demand matrix D.

Variables: Given a network topology, s_{ij} represents the weight of the static link from i to j, o_{ij} represents the weight of the reconfigurable link from i to j in the reconfigurable switch and D_{st} is the size of the demand from node s to node t.

We denote a matching from node i to j in the reconfigurable switch by setting the value of m_{ij} to 1. The boolean x_{ij}^{st} is set to 1 if a link from i to j is used in the shortest path from s to t, as well as y_{ij}^{st} if that link is a reconfigurable one. Finally, the length of the shortest path from s to t is given by $dist_{st}$.

Objective: The goal is to minimize the length of the shortest path for each communicating pair according to their priority.

$$min \sum_{s} \sum_{t} D_{st} dist_{st} \tag{2}$$

Constraints: A node connected to the reconfigurable switch can only have one incoming and one outgoing reconfigurable link in the directed case (3). In the bidirected case, creating a link from i to j also always creates the reverse link from j to i (4).

$$\sum_{j=1}^{n} m_{ij} \leq 1; \sum_{j=1}^{n} m_{ji} \leq 1 \tag{3}$$

$$m_{ij} = m_{ji} \tag{4}$$

Flow conservation: A flow that enters a node must leave it, with the exception of the start and end nodes.

$$\sum_j x_{ij}^{st} - \sum_j x_{ji}^{st} = \begin{cases} 1, & \text{if } i = s. \\ -1, & \text{if } i = t. \\ 0, & \text{otherwise.} \end{cases} \tag{5}$$

Path cost: The length of the path from the sender to the receiver is the sum of every link that is taken along the path. If a matching has occurred between two nodes, the length of the link between the two nodes is now the length of the reconfigurable link instead of the length of the static link.

$$dist_{st} = \sum_{i=1}^{n} \sum_{j=1}^{n} (s_{ij}(x_{ij}^{st} - y_{ij}^{st}) + o_{ij} y_{ij}^{st}) \tag{6}$$

Matching: If a reconfigurable link is taken between i and j then there must be a matching in the reconfigurable switch.

$$y_{ij}^{st} \leq m_{ij} \tag{7a}$$

$$y_{ij}^{st} \leq x_{ij}^{st} \tag{7b}$$

Even though the ILP presented in this section will achieve optimal results, its runtime is non-polynomial and it will not scale for larger instances. We thus present several polynomial-time heuristics, which we will evaluate in Section 5.

4.2. ReconfigDijkstra: A subroutine for a single flow

We start with the case of a single flow, which we will call as a subroutine in the later heuristics. We distinguish two cases, differentiating the reconfigurable link types.

The first is ReconfigDijkstra for directed reconfigurable links, *i.e.,* nodes connected to a reconfigurable switch can choose one outgoing and one incoming link. Therefore, we can use Dijkstra's algorithm [22] on a graph containing every possible matching candidate link and the static graph, updating it afterwards, simplifying the flow algorithm in [14].

For the case of undirected reconfigurable links we adapt the ReconfigDijkstra algorithm. We run it in the same fashion, on a graph in which every possible matching candidate link is present. The difference with this heuristic is that when it reaches a node via a reconfigurable link, it will not update new neighbors that are only visible through another reconfigurable link. This heuristic is optimal in the case where the reconfigurable links follow the triangular inequality [14]: if a shortest path requires that two reconfigurable hops (u, v), (v, w) to be taken one after the other then $|u, v| + |v, w| < |u, w|$ and the triangular inequality is violated. In other words, when e.g. all reconfigurable links have the same weight, we can run an analogous algorithm as in the directed version. In the case where the triangular inequality is not respected by the weights, the algorithm will give a correct solution, but it might not be optimal. It is an open question if the general case can be solved optimally in polynomial time [14]. To update the graph for future computations, we first transform the candidate links taken in the path into static links, and then delete the candidate links that become illegal. The reconfiguration step is linear as we are checking for every node its outgoing and incoming links involved in the matching. If implemented naïvely, the complexity is $O(n^2)$, which can be improved by using Fibonacci heaps in the ReconfigDijkstra algorithm [23].

4.3. DemandFirst: Large demands first

We first present DemandFirst in Algorithm 1. This algorithm greedily chooses the biggest demand, and routes it through the network, creating the best matches for it. It stops when all possible matches have been created or every communicating pair has been processed. As the algorithm consists of running ReconfigDijkstra for every demand, we find its complexity to be $O(dn^2)$ with d being the amount of non-zero entries in the demand matrix and n being the number of nodes in the network. While it is the fastest solution, it is oblivious to the interplay of different demands.

4.4. GainDemand: Gains and demands

A first improvement over DemandFirst is to take into consideration the impact that creating an optimal matching for a node-pair has on other pairs.

To this end we introduce the algorithm GainDemand (Algorithm 2). We compute, for every demand, the improvement of the matching (w.r.t. the objective value) created by the algorithm ReconfigDijkstra, storing it in an ordered list. More formally, this improvement is defined as:

Definition 1. Let N be a weighted hybrid network. The *gain* of a matching configuration $N(\mathcal{M})$ for a demand matrix D is the improvement in comparison just using the static network:

$$\sum_{(i,j)\in D} D[i,j] \cdot dist_{N(\emptyset)}(i,j) - \sum_{(i,j)\in D} D[i,j] \cdot dist_{N(\mathcal{M})}(i,j) \ .$$

We then run ReconfigDijkstra iteratively on the ordered list, until no more matching links can be created.

The complexity of this algorithm is $O(d^2n^2)$ as it finds the d paths for each of the d matching configurations. For further runtime improvements, it is possible to only consider the demands that will be affected by the creation of matching links. GainDemand finds its limitation when larger sets of communicating pairs are helpful to one another.

4.5. GainUpdate: Greedy paths

We next present algorithm GainUpdate (see Algorithm 3). It is inspired by GainDemand, but it recomputes the gain after every matching that occurs, in order to benefit from the current situation. In other words, when a set of demands creates a high gain for themselves, once one of these demand is routed, the gain can be much smaller at the next iteration. Its complexity is $O(d^3n^2)$ as we are executing a similar routine as in GainDemand after every demand gets routed.

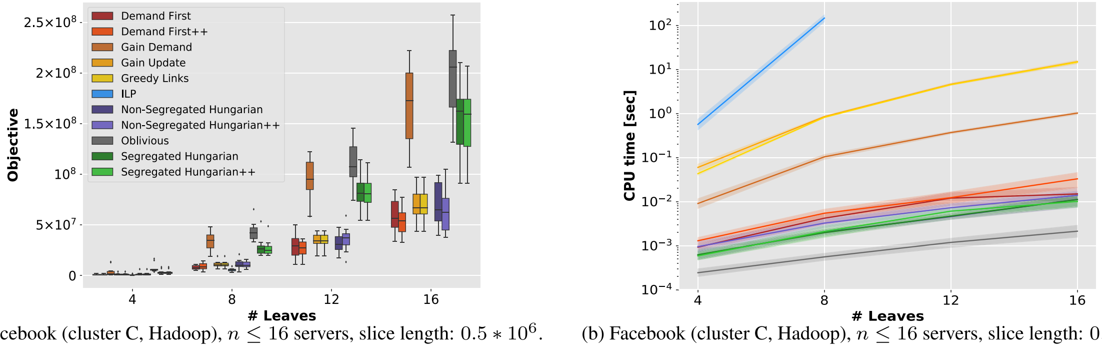
4.6. GreedyLinks: Greedy links

Lastly, in order to further shrink the gap to the optimal solution, we introduce GreedyLinks (see Algorithm 4). The principle is the same as in GainUpdate, but rather than considering the set of links introduced by a demand, we consider every possible link at each step. In terms of complexity it is similar to the previous algorithm on denser graphs, but much heavier on sparser graphs. The implementation is in $O(dn^4)$ for $d > 0$. Instead of executing the GainDemand routine on every demand, we will execute it on every possible reconfigurable link, which might form a complete graph.

4.7. Routing and further extensions

Note that the Algorithms 1 to 4 only provide the graph with the chosen reconfigurable links as an output, not the routing itself. To this end, we run the classic (Kleene–Roy–)Floyd–Warshall(–Ingerman)-algorithm [24, §9.8] to obtain All-Pairs-Shortest-Paths (APSP) in $O(n^3)$. Some slightly faster APSP-algorithms exist in special cases, see [25,26]. Moreover, as a heuristic speed-up, we can terminate the Algorithms 1 to 4 earlier if no more new reconfigurable links can be added.

As a further variant, we propose DemandFirst++, which takes the original path lengths in the static network into account, inspired by [14, §3.1]: When sorting the demand entries in Algorithm 1 in line 1, we first multiply each entry (i, j) in D with the distance between i and j in the static network, obtained by running APSP on the static network as well.



(a) Facebook (cluster C, Hadoop), $n \leq 16$ servers, slice length: $0.5 * 10^6$.

(b) Facebook (cluster C, Hadoop), $n \leq 16$ servers, slice length: $0.5 * 10^6$.

Fig. 3. Left: Comparison of the achieved objective values (Eq. (1)) for our different algorithms, the baseline approaches, and the ILP. Smaller objective values are better, as they represent the (weighted) average path lengths. Right: CPU time (*logarithmic scale*), where the shaded areas depict minimum and maximum values.

```
Algorithm 1: DemandFirst
Input: A weighted hybrid network $N$ and traffic demands $D$.
  1. Sort the demand entries in $D$ by size in descending order.
  2. Until the list is empty or no more links can be created do:
      (a) Pick the first entry $D_{i,j}$ in $D$.
      (b) Run ReconfigDijkstra for $i$,$j$ on $N$.
      (c) Update $N$, delete the first entry in $D$.
Output: A graph $N$ with the newly created links.
```

Algorithm 1: Pseudocode of the DemandFirst algorithm.

```
Algorithm 2: GainDemand
Input: A weighted hybrid network $N$ and traffic demands $D$.
  1. Initialize an empty list value.
  2. For every entry $(i,j)$ in $D$:
      (a) Run ReconfigDijkstra on $N$, adding the value of the
          objective function in the updated $N$ to value.
      (b) Reset $N$ to its initial state.
  3. Sort the items in value by size.
  4. For the values of $(i,j)$ in value in descending order, until empty or all
     possible matching links have been created:
      (a) Run ReconfigDijkstra for $(i,j)$ on $N$.
      (b) Update $N$, delete the first entry in $value$.
Output: A graph $N$ with the newly created links.
```

Algorithm 2: Pseudocode of the GainDemand algorithm.

```
Algorithm 3: GainUpdate
Input: A weighted hybrid network $N$ and traffic demands $D$.
  1. Initialize an empty list value.
  2. Until the demand matrix $D$ has no more entries or all possible matching
     links have been created:
      (a) For every demand in $D$
           i. Run ReconfigDijkstra on $N$, adding the value of the
              objective function in the updated $N$ to value.
          ii. Reset $N$ to its initial state.
      (b) Find the maximum entry $(i,j)$ in value.
      (c) Run ReconfigDijkstra for $(i,j)$ in $D$, update $N$.
      (d) Remove the entry $(i,j)$ from $D$ and clear value.
Output: A graph $N$ with the newly created links.
```

Algorithm 3: Pseudocode of the GainUpdate algorithm.

```
Algorithm 4: GreedyLinks
Input: A weighted hybrid network $N$ and traffic demands $D$.
  1. Initialize an empty list value.
  2. Until no more matching links can be added to $N$:
      (a) For every possible further reconfigurable link $e$ in $N$:
            i. Add $e$ as a matching link to $N$.
           ii. Compute the value of the objective function on $N$.
          iii. Add the value (for the link $e$) to value.
           iv. Reset $N$ by removing $e$ as a matching link.
      (b) Find the best link $e$ in value.
      (c) Update $N$ with the reconfigurable link $e$.
      (d) Clear the list value.
Output: A graph $N$ with the newly created links.
```

Algorithm 4: Pseudocode of the GreedyLinks algorithm.

5. Evaluation

In order to assess the benefits of non-segregated routing and joint optimization of the reconfigurable network, and to compare the different algorithms introduced above, we conducted extensive simulations. We consider a datacenter scenario and use real-world workloads based on Facebook's datacenter traffic data [27–29], along with traffic matrices from production clusters at Microsoft [30], from high performance computing [31], and pFabric [32], see also [33] for a structural analysis.

5.1. Methodology

The trace data of these sources is stored in a noSql database (MongoDB), where each independent set of connection pairs is stored in a separate collection. In order to generate workloads, we use the above described sources of trace data. Herein the Microsoft ProjecToR traffic matrix data is leveraged to create traces by respecting the communication probabilities between ToRs, using Python's default random number generator.

Tests are performed multiple times with different slices of the request sequence. The samples are generated by defining 10 different equally distributed starting points of the total request sequence. To scale the problem size, we filter a slice by n pseudo randomly selected servers. To provide a dense demand we accumulate the requests of the selected servers within this slice of the sequence. We map the corresponding demand to the leaves (*i.e.,* the servers) of a three tier k-ary fat tree [2] of diameter 6: the fixed network hence describes a Clos topology. In addition to the Clos network, the servers can be connected via a single optical circuit switch providing a matching. In order to preserve locality, we order the leaves according to their IP addresses. The weights of the directed optical links are set to 1, while the static link weights are set to 5.

In order to ensure a high density of the demand and hence render the experiments more interesting, we increased the size of the request slices to an extent, where a further increase would not lead to a density gain. Furthermore, to increase the density of the demand we interpret the rack-to-rack demands of the facebook traces as server-to-server demands, between the leaves. Intra-rack communication is hence ignored.

For our experiments, trace data was stored in MongoDB and simulations executed on a HP DL380 G9 with 2 Intel Xeons E5-2697V3 SR1XF with 2.6 GHz, 14 cores[1] each and a total of 128 GB DDR4 RAM. The host machine was running Ubuntu 18.04.3 LTS. We implemented the proposed algorithm in Python (3.7.4) [34] leveraging the NetworkX library (2.3) [35]. To solve the ILP we used Gurobi (8.1.1) [36]. The Hungarian matching algorithm is based on Munkres (1.0.7) [37].

We set a runtime limit of 180 min for each test, which limits the completion of the ILP for larger instances, but is irrelevant for the faster heuristics, except for the ones recomputing iteratively based on gain, which run into problems in larger networks. We note that the gain algorithms are not parallelized (the same holds for all heuristics), they only run on a single core. Furthermore, their implementation could benefit from improved gain re-computation and a port to *e.g.,* C++. However, as our focus is on the *quality* of the achieved solutions, we defer such optimizations to future work.

5.2. Baseline

Even though prior work [14] showed that non-segregated routing improves over (artificially) segregated routing, the level of improvement was not studied yet. To this end, we use standard[2] approaches for segregated routing in hybrid networks as a baseline, described next.

Recall that only the leaf nodes act as sources and destinations in our setting. We hence compute a maximum weight bipartite matching between the outgoing and incoming reconfigurable ports, as suggested in *e.g., Helios* [16], denoted as `SegregatedHungarian`. The routing is then performed in a segregated manner, where a route may either use a direct matching link or the static network parts.

5.3. Further algorithms

Analogously as in Section 4.7, we implement ++ versions of both segregated matching algorithms, where, only for the matching computation, each demand entry is multiplied by its distance in the static network.

For the sake of completeness, we furthermore include the performance without the optical circuit switch, implemented via APSP in NetworkX, denoted as `Oblivious`.

Moreover, we also lift the segregated algorithms to non-segregated routing, by employing routing via APSP as well.

5.4. Results

We first describe our results on very small networks with 4–16 nodes, then medium size networks up to 121–432 nodes, followed by large networks with up to 1024 nodes

Small networks. For the smallest network size in Fig. 3, with 4 to 16 nodes, we use traces from Facebook's cluster C (Hadoop).

Performance: As shown in Fig. 3(a), the ILP achieves the best objective values, but nearly all other non-segregated algorithms achieve similar path lengths in the very small networks. Only `GainDemand` is an outlier, it performs even worse than the segregated algorithms. Moreover, we can already on these small networks see a significant difference when the optical circuit switch is turned off, roughly a factor of 2 to 4.

CPU time: Regarding the CPU time in Fig. 3(b), the ILP's usage is excessive, multiple magnitudes beyond the other algorithms. `GainDemand`, `GainUpdate`, and `GreedyLinks` are also at least 10–100 times slower than the remaining algorithms. The other algorithms have similar CPU times, with the segregated ones being a bit faster, but still about 10 times slower than `Oblivious`.

Medium size networks: Facebook cluster B (web) traces. The evaluation results from Facebook's cluster B traces, for 16 to 250 nodes, are presented in Figs. 4(a) and 4(b).

Performance: As can be seen in Fig. 4(a), `Oblivious`, `GainDemand`, and the segregated matching algorithms perform roughly similar, but all having the worst performance. The non-segregated matching and greedy algorithms beat them by a factor of roughly 1.5, with the `DemandFirst` variants performing even better. At 250 nodes, the achieved objective value is roughly 10% less, with the ++ versions marginally ahead.

CPU time: The CPU time in Fig. 4(b) shows a similar trend as in the smaller networks in Fig. 3(b). `GreedyLinks` and `GainUpdate` are more than 100 to 1000 times slower than `DemandFirst`, with `GainDemand` being about 10 to 100 times slower. The matching algorithms are up to 5 times faster than `DemandFirst` for 16 nodes (`DemandFirst` still finishes in less than a second), but the run times converge for larger networks, being roughly identical for 250 nodes. `Oblivious` is always faster and the gap increases with the network size, from about 10 to a bit over 100 times for larger networks.

Medium size networks: High performance computing traces. Our high performance computing traces, where we have data for up to 128 nodes, are shown in Figs. 4(c) and 4(d).

Performance: The situation is similar to before, where in Fig. 4(c) `GainDemand` and the segregated matchings perform worst of all algorithms using the optical circuit switch, only beating `Oblivious`. Of the other algorithms, `DemandFirst` variants, `GainUpdate`, and `GreedyLinks` perform similarly and best, slightly ahead of the non-segregated matchings. The performance gap between `DemandFirst` and `SegregatedHungarian` variants is still significant (1.5–2×).

CPU time: Regarding CPU time, the situation in Fig. 4(d) is analogous to Fig. 4(b), when considering 16 to 128 nodes, though we slightly more variations in each algorithm's run time.

Medium size networks: Microsoft ProjecToR traces. The results for the traces from Microsoft are shown in Figs. 4(e) and 4(f), where the evaluated data set contains up to 121 nodes.

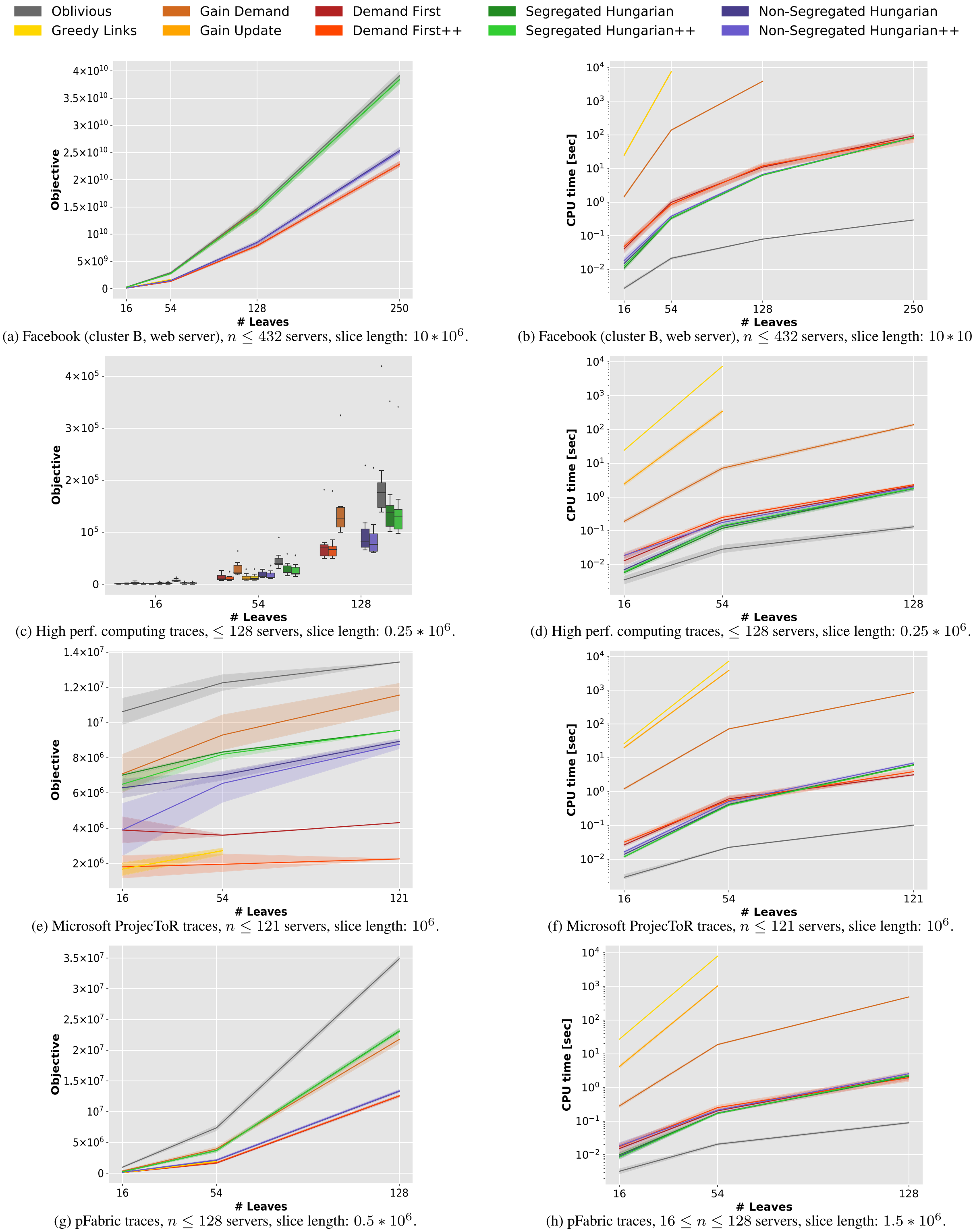
Performance: Whereas all other medium size trace evaluations are roughly similar on a course scale, the Microsoft traces in particular highlight the differences to the ++ versions. `DemandFirst` is about 2 times worse than its ++ variant, for all network sizes. For 121 nodes, all further algorithms are at least 2 (4) times slower than `DemandFirst` (++) too, though the performance of `NonSegregatedHungarian` matches `DemandFirst` for 16 nodes. The segregated matching trails a bit behind the non-segregated variants, where in turn again `GainDemand` and `Oblivious` fall behind further. The performance of `GreedyLinks` and `GainUpdate` is between `DemandFirst` and its ++ variant.

CPU time: In Fig. 4(f) `GreedyLinks`, `GainUpdate`, and `GainDemand` are again several orders of magnitude slower than all other algorithms, whereas `Oblivious` is about 10 to 100 times faster than the rest. However, all these remaining algorithms use roughly similar CPU times, with `DemandFirst` being slightly slower for smaller sizes and again being slightly faster for the largest network size.

Medium size networks: pFabric traces. The evaluation results for the pFabric traces are shown in Figs. 4(g) and 4(h).

[1] However, each algorithm (except the ILP) only utilized a single core.

[2] Standard with respect to the studied topology.



(a) Facebook (cluster B, web server), $n \leq 432$ servers, slice length: $10 * 10^6$.

(b) Facebook (cluster B, web server), $n \leq 432$ servers, slice length: $10 * 10^6$.

(c) High perf. computing traces, ≤ 128 servers, slice length: $0.25 * 10^6$.

(d) High perf. computing traces, ≤ 128 servers, slice length: $0.25 * 10^6$.

(e) Microsoft ProjecToR traces, $n \leq 121$ servers, slice length: 10^6.

(f) Microsoft ProjecToR traces, $n \leq 121$ servers, slice length: 10^6.

(g) pFabric traces, $n \leq 128$ servers, slice length: $0.5 * 10^6$.

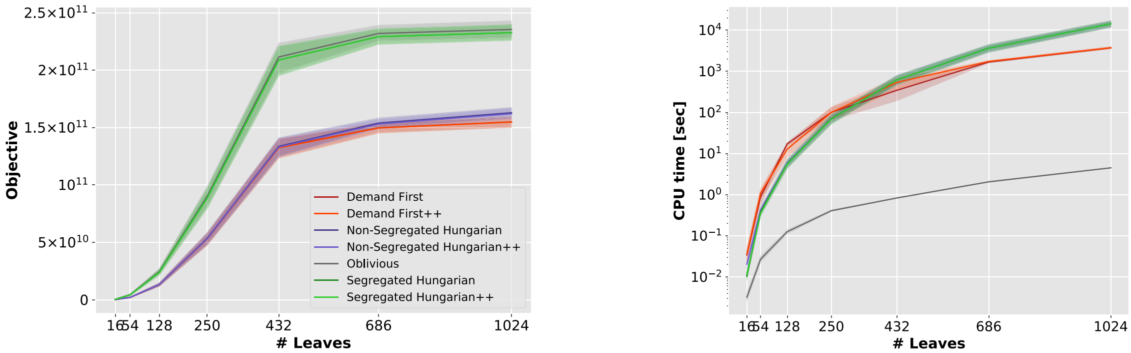
(h) pFabric traces, $16 \leq n \leq 128$ servers, slice length: $1.5 * 10^6$.

Fig. 4. Comparison of the different algorithms regarding objective value and CPU time.

Performance: As expected, Oblivious again performs worst in Fig. 4(g), about 1.5× behind the segregated matching algorithms and GainDemand, which in turn are about 1.5× behind the remaining non-segregated algorithms. From this group, the DemandFirst variants are slightly better than the NonSegregatedHungarian, with GreedyLinks being between both the latter and DemandFirst.

CPU time: The CPU time comparison in Fig. 4(h) is roughly analogous to the high performance computing evaluations.

Large size networks. For the largest network size, ranging from 16 to 1024 nodes, we consider traces from Facebook cluster C (Hadoop) in Fig. 5, as it is the only trace sample we have with sufficiently many nodes. Herein we only evaluate the fastest and best performing



(a) Facebook (cluster C, Hadoop), $n \leq 1024$ servers, slice length: 10^6.

(b) Facebook (cluster C, Hadoop), $n \leq 1024$ servers, slice length: 10^6.

Fig. 5. Objective (left) and CPU time (right) values for fast algorithms in larger directed networks.

algorithms from earlier experiments, i.e., DemandFirst, SegregatedHungarian (also non-segregated), also their ++ variants, and Oblivious.

Performance: The achieved objective value in Fig. 5(a) shows a clear trend of two classes of algorithms, (1) non-segregated and (2) segregated respectively Oblivious. Herein the non-segregated versions perform roughly 2 times better than the other algorithms until 432 nodes, followed by a slightly lesser performance increase of about 1.5 times after. For the latter, the remaining demands in the traces are more sparse, making optimization less effective. DemandFirst slightly outperforms NonSegregatedHungarian, where each time the ++ variants add a barely visible level of improvement.

CPU time: For the CPU time in Fig. 5(b), the matching algorithms are a bit faster than the DemandFirst variants, where the trend is reversed from 432 nodes on, where for 1024 nodes, DemandFirst uses only about 50% of the CPU time in comparison. For the matching algorithms, the non-segregated ones are slightly slower for smaller node sizes, but the run time nearly matches up for larger networks. The Oblivious curve is always the lowest, several magnitudes faster for large sizes.

5.5. Discussion and outlook

All in all, we can conclude that non-segregated approaches significantly outperform the standard segregated routing methods on hybrid networks, for all evaluated settings from the four traffic trace sets. Additionally, for all studied algorithms, ++ variants improve performance slightly (significantly for the Microsoft traces) and are hence preferable. Interestingly, the simplest of our heuristics, DemandFirst, in its ++ variants, typically also provides the best results, making it an attractive solution in practice: its runtime is significantly lower than that of the also well-performing GreedyLinks, and about the same as for both variants of NonSegregatedHungarian, and the provided route lengths are shorter.

Notwithstanding, even though our DemandFirst approach is not slower than the matching algorithms, the runtime still grows rapidly with network size. For medium-sized networks, optimized implementations in low-level languages opposed to in Python, or even in FPGAs or ASICs, could bring the runtime down to sub-second scale, possibly augmented by utilizing our algorithms to train machine-learning methods [38,39]. For larger networks however, we believe fundamentally different ideas are needed.

As a first step, DemandFirst could leverage centralized dynamic graph algorithms to compute short paths [40,41], as the network only changes slightly in each iteration when adding a flow. Conceptually, this could also be extended to the setting where the traffic matrix does not change much, and hence prior solutions can be leveraged, also for matchings [42]. As another speedup, parallel algorithms could be employed, as already explored for (approximate) matchings in MapReduce environments [43], shared-memory multiprocessors [44], and supercomputers [45].

To scale further however, distributed algorithms with local knowledge might be the best direction. There are first intriguing proposals, which however are also limited. *ProjecToR* [9] uses a stable-marriage idea with distributed proposals, but on the other hand considers only segregated routing along single hops. *DiSplayNet* [46] instead utilizes non-segregated routing, but then only builds tree networks. To go beyond, we envision different avenues to build fully flexible demand-aware networks in a distributed fashion.

First, and maybe most straightforward, is to implement distributed (approximate) matching algorithms [47, §4.3], where each node is updated with its own traffic demands. A second approach would be to extend e.g. DemandFirst to so-called hybrid distributed computing models, with fast local and congested global communication. Interestingly and fitting, work in this model so far focused on shortest path computations [48–50]. Third, and so far most unexplored, would be to extend the earlier mentioned centralized dynamic graph algorithms to a distributed setting [51], also via distributed control planes [52] and preprocessing [53,54]. In these contexts, so far only considered non-distributed, it would be worthwhile to investigate the flexibility [55,56] of network designs and the perspective of online algorithms [57].

Lastly, and orthogonal to the above directions, recent work observed that some datacenter traffic exhibits weak temporal stability over hours to days, which leads to a "*slow-varying clustering effect*" [58]. In such settings, even long precomputation times would be worthwhile, with a small performance trade-off in comparison to ongoing reconfiguration. Still, such stability heavily depends on the specific scenarios [33] and can hence not always be assumed.

6. Related work

The benefits and limitations of reconfigurable (hybrid) networks, which not only arise in datacenters but also in wide-area networks [59–67], are currently discussed intensively in the literature, see e.g., [8,9,16,17,59,68–72] and the survey in [73]. There is a wide spectrum of approaches to make datacenter topologies more dynamic, with solutions ranging from approaches leveraging converter switches to dynamically change between a Clos network and approximate random graphs [74] to approaches based on rotor switches rotating through a set of pre-defined matchings [75,76], as well as optical multicast [77–79]. Some empirical studies have shown that depending on the workload, demand-aware networks can achieve a performance similar to demand-oblivious networks at lower cost [8,9].

Less is known about the underlying algorithmic problem of designing and routing on such topologies. The problem is related to *graph augmentation* [80,81] literature considering how to enhance a given (fixed) graph with an optimal number of "extra edges", sometimes also referred to as "ghost edges" [82]: the objective in this literature is typically to provide small world properties [83] or minimize the network diameter [84,85]. However, most of these algorithms are not applicable directly here, where rather than individual edges, entire matchings are added.

In this context, it could also be interesting to consider the removal of links, *e.g.,* for the scheduling of link repairs [86].

Most existing algorithms on the optimization of reconfigurable topologies are restricted to "segregated" routing models where traffic is forced to either use the fixed network or a *single* reconfigurable link (see e.g., [9,14]), and consider simple heuristics based on matchings [8,15–18] and related concepts, such as edge-coloring [87] and stable-marriage schemes [9]. The closest paper to ours is the work by Foerster et al. [14,88], which we extend by presenting several efficient algorithms for non-segregated routing which we also evaluate in simulations. Furthermore, we show that not only computing exact solutions is NP-hard, but also computing approximations.

Finally we note that there also exists much recent work on non-hybrid, fully reconfigurable (static and dynamic) topologies that do not account for the possibility of oblivious (fixed) links [10,46,89–92].

7. Conclusion

This paper initiated the study of efficient algorithms for non-segregated routing and optimization of emerging reconfigurable network topologies. We have shown that while the underlying problem is hard to approximate in the worst-case, fast and simple algorithms can significantly improve the performance compared to the state-of-the-art, which is also confirmed by our trace-driven simulations.

We understand our work as a first step and believe that it opens several interesting avenues for future research. In particular, it will be interesting to provide a more complete picture of upper and lower bounds on approximations, also by considering randomized routing.

Moreover, it would also be interesting to consider dynamic or online settings [10], possibly in conjunction with consistent network updates [93].

Reproducibility

In order to simplify future research and in order to make our results reproducible, we created a public code repository at https://gitlab.cs.univie.ac.at/ct-papers/2020-non-segregated-reconfigurable-routing.

Bibliographical note

A preliminary version of this article appeared in the conference proceedings of IFIP Networking 2019 [94].

CRediT authorship contribution statement

Thomas Fenz: Methodology, Formal analysis, Software, Validation, Visualization. **Klaus-Tycho Foerster:** Conceptualization, Methodology, Formal analysis, Supervision. **Stefan Schmid:** Conceptualization, Funding acquisition, Resources, Project administration. **Anaïs Villedieu:** Methodology, Formal analysis, Software, Validation.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

We thank Andreas Blenk and Johannes Zerwas for several discussions that helped to improve this paper. We also thank the anonymous reviewers of IFIP Networking 2019 and Elsevier Computer Communications for their valuable feedback. We moreover thank Esra Ceylan for her comments that improved the presentation of this paper. This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement 864228).

References

[1] M. Noormohammadpour, C.S. Raghavendra, Datacenter traffic control: Understanding techniques and tradeoffs, IEEE Commun. Surv. Tutor. 20 (2) (2017) 1492–1525.
[2] M. Al-Fares, A. Loukissas, A. Vahdat, A scalable, commodity data center network architecture, in: SIGCOMM, ACM, 2008.
[3] S. Kassing, A. Valadarsky, G. Shahaf, M. Schapira, A. Singla, Beyond fat-trees without antennae, mirrors, and disco-balls, in: SIGCOMM, ACM, 2017.
[4] C. Guo, G. Lu, D. Li, H. Wu, X. Zhang, Y. Shi, C. Tian, Y. Zhang, S. Lu, Bcube: a high performance, server-centric network architecture for modular data centers, in: SIGCOMM, ACM, 2009.
[5] A. Singla, C. Hong, L. Popa, P.B. Godfrey, Jellyfish: Networking data centers randomly, in: NSDI, USENIX, 2012.
[6] A.G. Greenberg, J.R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D.A. Maltz, P. Patel, S. Sengupta, VL2: a scalable and flexible data center network, in: SIGCOMM, ACM, 2009.
[7] V. Liu, D. Halperin, A. Krishnamurthy, T.E. Anderson, F10: a fault-tolerant engineered network, in: NSDI, USENIX, 2013.
[8] N.H. Azimi, Z.A. Qazi, H. Gupta, V. Sekar, S.R. Das, J.P. Longtin, H. Shah, A. Tanwer, Firefly: a reconfigurable wireless data center fabric using free-space optics, in: SIGCOMM, ACM, 2014.
[9] M. Ghobadi, R. Mahajan, A. Phanishayee, N.R. Devanur, J. Kulkarni, G. Ranade, P. Blanche, H. Rastegarfar, M. Glick, D.C. Kilper, ProjecToR: Agile reconfigurable data center interconnect, in: SIGCOMM, ACM, 2016.
[10] C. Avin, S. Schmid, Toward demand-aware networking: a theory for self-adjusting networks, ACM SIGCOMM Comput. Commun. Rev. 48 (5) (2018) 31–40.
[11] M. Alizadeh, A.G. Greenberg, D.A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, M. Sridharan, Data center TCP (DCTCP), in: SIGCOMM, ACM, 2010.
[12] A. Roy, H. Zeng, J. Bagga, G. Porter, A.C. Snoeren, Inside the social network's (datacenter) network, in: SIGCOMM, ACM, 2015.
[13] G. Judd, Attaining the promise and avoiding the pitfalls of TCP in the datacenter, in: NSDI, USENIX, 2015, pp. 145–157.
[14] K.-T. Foerster, M. Ghobadi, S. Schmid, Characterizing the algorithmic complexity of reconfigurable data center architectures, in: ANCS, IEEE/ACM, 2018.
[15] G. Wang, D.G. Andersen, M. Kaminsky, K. Papagiannaki, T.S.E. Ng, M. Kozuch, M.P. Ryan, c-Through: part-time optics in data centers, in: SIGCOMM, 2010.
[16] N. Farrington, G. Porter, S. Radhakrishnan, H.H. Bazzaz, V. Subramanya, Y. Fainman, G. Papen, A. Vahdat, Helios: a hybrid electrical/optical switch architecture for modular data centers, in: SIGCOMM, ACM, 2010.
[17] H. Liu, F. Lu, A. Forencich, R. Kapoor, M. Tewari, G.M. Voelker, G. Papen, A.C. Snoeren, G. Porter, Circuit switching under the radar with REACToR, in: NSDI, USENIX, 2014.
[18] H. Liu, M.K. Mukerjee, C. Li, N. Feltman, G. Papen, S. Savage, S. Seshan, G.M. Voelker, D.G. Andersen, M. Kaminsky, G. Porter, A.C. Snoeren, Scheduling techniques for hybrid circuit/packet networks, in: CoNEXT, ACM, 2015.
[19] S.B. Venkatakrishnan, M. Alizadeh, P. Viswanath, Costly circuits, submodular schedules and approximate Carathéodory theorems, in: SIGMETRICS, ACM, 2016.
[20] U. Feige, A threshold of ln n for approximating set cover, J. ACM 45 (4) (1998).
[21] P. Crescenzi, A short guide to approximation preserving reductions, in: IEEE Conference on Computational Complexity, 1997.
[22] E.W. Dijkstra, A note on two problems in connexion with graphs, Numer. Math. 1 (1) (1959) 269–271.
[23] M.L. Fredman, R.E. Tarjan, Fibonacci heaps and their uses in improved network optimization algorithms, J. ACM 34 (3) (1987) 596–615.
[24] J. Erickson, Algorithms, 2019, https://jeffe.cs.illinois.edu/teaching/algorithms/.
[25] T.M. Chan, More algorithms for all-pairs shortest paths in weighted graphs, SIAM J. Comput. 39 (5) (2010) 2075–2089.
[26] R.R. Williams, Faster all-pairs shortest paths via circuit complexity, SIAM J. Comput. 47 (5) (2018) 1965–1985.
[27] A. Roy, H. Zeng, J. Bagga, G. Porter, A.C. Snoeren, Inside the social network's (datacenter) network, in: SIGCOMM, ACM, 2015.
[28] J.H. Zeng, Data sharing on traffic pattern inside Facebook's datacenter network, 2017, https://research.fb.com/data-sharing-on-traffic-pattern-inside-facebooks-datacenter-network/.

[29] facebook, Facebook network analytics data sharing, 2018, https://www.facebook.com/groups/1144031739005495/.

[30] M. Ghobadi, et al., ProjecToR: Agile reconfigurable data center interconnect, 2016, https://www.microsoft.com/en-us/research/project/projector-agile-reconfigurable-data-center-interconnect/.

[31] Characterization of the DOE mini-apps, 2016, https://portal.nersc.gov/project/CAL/doe-miniapps.htm.

[32] M. Alizadeh, S. Yang, M. Sharif, S. Katti, N. McKeown, B. Prabhakar, S. Shenker, PFabric: minimal near-optimal datacenter transport, in: SIGCOMM, ACM, 2013.

[33] C. Avin, M. Ghobadi, C. Griner, S. Schmid, On the complexity of traffic traces and implications, Proc. ACM Meas. Anal. Comput. Syst. 4 (1) (2020) 20:1–20:29.

[34] Python 3.7.4, 2019, www.python.org/downloads/release/python-374/.

[35] NetworkX, 2019, https://networkx.github.io/documentation/networkx-2.3/.

[36] Gurobi optimizer, 2019, www.gurobi.com/downloads/gurobi-software/.

[37] Munkres (hungarian) algorithm for the assignment problem, 2014, https://pypi.org/project/munkres/#history.

[38] A. Valadarsky, M. Schapira, D. Shahaf, A. Tamar, Learning to route, in: HotNets, ACM, 2017, pp. 185–191.

[39] S. Salman, C. Streiffer, H. Chen, T. Benson, A. Kadav, Deepconf: Automating data center network topologies management with machine learning, in: NetAI@SIGCOMM, ACM, 2018, pp. 8–14.

[40] M. Henzinger, S. Krinninger, D. Nanongkai, Decremental single-source shortest paths on undirected graphs in near-linear total update time, J. ACM 65 (6) (2018) 36:1–36:40.

[41] M. Henzinger, S. Krinninger, D. Nanongkai, Dynamic approximate all-pairs shortest paths: Breaking the O(mn) barrier and derandomization, SIAM J. Comput. 45 (3) (2016) 947–1006.

[42] S. Bhattacharya, D. Chakrabarty, M. Henzinger, Deterministic dynamic matching in O(1) update time, Algorithmica 82 (4) (2020) 1057–1080.

[43] F.M. Manshadi, B. Awerbuch, R. Gemulla, R. Khandekar, J. Mestre, M. Sozio, A distributed algorithm for large-scale generalized matching, Proc. VLDB Endow. 6 (9) (2013) 613–624.

[44] A.M. Khan, A. Pothen, M.M.A. Patwary, N.R. Satish, N. Sundaram, F. Manne, M. Halappanavar, P. Dubey, Efficient approximation algorithms for weighted b-matching, SIAM J. Sci. Comput. 38 (5) (2016).

[45] A.M. Khan, A. Pothen, M.M.A. Patwary, M. Halappanavar, N.R. Satish, N. Sundaram, P. Dubey, Designing scalable b-matching algorithms on distributed memory multiprocessors by approximation, in: SC, IEEE, 2016.

[46] B. Peres, O.A. de Oliveira Souza, O. Goussevskaia, C. Avin, S. Schmid, Distributed self-adjusting tree networks, in: INFOCOM, IEEE, 2019.

[47] J. Suomela, Survey of local algorithms, ACM Comput. Surv. 45 (2) (2013) 24:1–24:40.

[48] M. Feldmann, K. Hinnenthal, C. Scheideler, Fast hybrid network algorithms for shortest paths in sparse graphs, 2020, CoRR abs/2007.01191.

[49] J. Augustine, K. Hinnenthal, F. Kuhn, C. Scheideler, P. Schneider, Shortest paths in a hybrid network model, in: SODA, SIAM, 2020.

[50] F. Kuhn, P. Schneider, Computing shortest paths and diameter in the hybrid network model, 2020, CoRR abs/2006.08408.

[51] K.-T. Foerster, J.H. Korhonen, A. Paz, J. Rybicki, S. Schmid, Input-dynamic distributed graph algorithms for congested networks, 2020, CoRR abs/2005.07637.

[52] S. Schmid, J. Suomela, Exploiting locality in distributed sdn control, in: HotSDN, ACM, 2013.

[53] K.-T. Foerster, J. Hirvonen, S. Schmid, J. Suomela, On the power of preprocessing in decentralized network optimization, in: INFOCOM, IEEE, 2019.

[54] K.-T. Foerster, J.H. Korhonen, J. Rybicki, S. Schmid, Does preprocessing help under congestion? in: PODC, ACM, 2019.

[55] W. Kellerer, A. Basta, A. Blenk, Using a flexibility measure for network design space analysis of SDN and NFV, in: INFOCOM Workshops, IEEE, 2016.

[56] M. He, A.M. Alba, A. Basta, A. Blenk, W. Kellerer, Flexibility in softwarized networks: Classifications and research challenges, IEEE Commun. Surv. Tutor. 21 (3) (2019) 2600–2636.

[57] M. Bienkowski, D. Fuchssteiner, J. Marcinkowski, S. Schmid, A competitive B-matching algorithm for reconfigurable datacenter networks, 2020, CoRR abs/2006.10692.

[58] M.Y. Teh, S. Zhao, K. Bergman, METTEOR: robust multi-traffic topology engineering for commercial data center networks, 2020, CoRR abs/2002.00473.

[59] S. Jia, X. Jin, G. Ghasemiesfeh, J. Ding, J. Gao, Competitive analysis for online scheduling in software-defined optical WAN, in: INFOCOM, IEEE, 2017.

[60] X. Jin, Y. Li, D. Wei, S. Li, J. Gao, L. Xu, G. Li, W. Xu, J. Rexford, Optimizing bulk transfers with software-defined optical WAN, in: SIGCOMM, ACM, 2016.

[61] R. Singh, M. Ghobadi, K.-T. Foerster, M. Filer, P. Gill, RADWAN: Rate adaptive wide area network, in: SIGCOMM, ACM, 2018.

[62] R. Singh, M. Ghobadi, K.-T. Foerster, M. Filer, P. Gill, Run, walk, crawl: Towards dynamic link capacities, in: HotNets, ACM, 2017.

[63] L. Luo, K.-T. Foerster, S. Schmid, H. Yu, DaRTree: deadline-aware multicast transfers in reconfigurable wide-area networks, in: IWQoS, ACM, 2019.

[64] K.-T. Foerster, L. Luo, M. Ghobadi, Optflow: A flow-based abstraction for programmable topologies, in: SOSR, ACM, 2020.

[65] J. Gossels, G. Choudhury, J. Rexford, Robust network design for IP/optical backbones, J. Opt. Commun. Netw. 11 (8) (2019) 478–490.

[66] L. Luo, K.-T. Foerster, S. Schmid, H. Yu, Deadline-aware multicast transfers in software-defined optical wide-area networks, IEEE J. Sel. Areas Commun. (2020).

[67] M. Dinitz, B. Moseley, Scheduling for weighted flow and completion times in reconfigurable networks, in: INFOCOM, IEEE, 2020.

[68] W. Dai, K.-T. Foerster, D. Fuchssteiner, S. Schmid, Load-optimization in reconfigurable networks: Algorithms and complexity of flow routing, in: Proc. 38th International Symposium on Computer Performance, Modeling, Measurements and Evaluation, PERFORMANCE, 2020.

[69] A. Singla, A. Singh, K. Ramachandran, L. Xu, Y. Zhang, Proteus: a topology malleable data center network, in: HotNets, ACM, 2010.

[70] D. Halperin, S. Kandula, J. Padhye, P. Bahl, D. Wetherall, Augmenting data center networks with multi-gigabit wireless links, in: SIGCOMM, ACM, 2011.

[71] X. Zhou, Z. Zhang, Y. Zhu, Y. Li, S. Kumar, A. Vahdat, B.Y. Zhao, H. Zheng, Mirror mirror on the ceiling: Flexible wireless links for data centers, ACM SIGCOMM Comput. Commun. Rev. 42 (4) (2012) 443–454.

[72] K. Chen, A. Singla, A. Singh, K. Ramachandran, L. Xu, Y. Zhang, X. Wen, Y. Chen, OSA: an optical switching architecture for data center networks with unprecedented flexibility, IEEE/ACM Trans. Netw. 22 (2) (2014) 498–511.

[73] K.-T. Foerster, S. Schmid, Survey of reconfigurable data center networks: Enablers, algorithms, complexity, SIGACT News 50 (2) (2019) 62–79.

[74] Y. Xia, X.S. Sun, S. Dzinamarira, D. Wu, X.S. Huang, T.S.E. Ng, A tale of two topologies: Exploring convertible data center network architectures with flat-tree, in: SIGCOMM, ACM, 2017.

[75] W.M. Mellette, R. McGuinness, A. Roy, A. Forencich, G. Papen, A.C. Snoeren, G. Porter, Rotornet: A scalable, low-complexity, optical datacenter network, in: SIGCOMM, ACM, 2017.

[76] W.M. Mellette, R. Das, Y. Guo, R. McGuinness, A.C. Snoeren, G. Porter, Expanding across time to deliver bandwidth efficiency and low latency, in: NSDI, 2020.

[77] X.S. Sun, T. Ng, When creek meets river: Exploiting high-bandwidth circuit switch in scheduling multicast data, in: ICNP, IEEE, 2017.

[78] Y. Xia, T. Ng, X.S. Sun, Blast: Accelerating high-performance data analytics applications by optical multicast, in: INFOCOM, IEEE, 2015.

[79] L. Luo, K.-T. Foerster, S. Schmid, H. Yu, Splitcast: Optimizing multicast flows in reconfigurable datacenter networks, in: INFOCOM, IEEE, 2020.

[80] A. Gozzard, M. Ward, A. Datta, Converting a network into a small-world network: Fast algorithms for minimizing average path length through link addition, Inform. Sci. 422 (2018).

[81] A. Meyerson, B. Tagiku, Minimizing average shortest path distances via shortcut edge addition, in: APPROX/RANDOM, Springer, 2009.

[82] M. Papagelis, F. Bonchi, A. Gionis, Suggesting ghost edges for a smaller world, in: CIKM, ACM, 2011, pp. 2305–2308.

[83] N. Parotsidis, E. Pitoura, P. Tsaparas, Selecting shortcuts for a smaller world, in: SDM, SIAM, 2015, pp. 28–36.

[84] D. Bilò, L. Gualà, G. Proietti, Improved approximability and non-approximability results for graph diameter decreasing problems, Theoret. Comput. Sci. 417 (2012) 12–22.

[85] E.D. Demaine, M. Zadimoghaddam, Minimizing the diameter of a network using shortcut edges, in: SWAT, 2010.

[86] D. Zhuo, M. Ghobadi, R. Mahajan, K.-T. Foerster, A. Krishnamurthy, T.E. Anderson, Understanding and mitigating packet corruption in data center networks, in: SIGCOMM, ACM, 2017.

[87] L. Chen, K. Chen, Z. Zhu, M. Yu, G. Porter, C. Qiao, S. Zhong, Enabling wide-spread communications on optical fabric with megaswitch, in: NSDI, 2017.

[88] K.-T. Foerster, M. Pacut, S. Schmid, On the complexity of non-segregated routing in reconfigurable data center architectures, Comput. Commun. Rev. 49 (2) (2019) 2–8.

[89] C. Avin, A. Hercules, A. Loukas, S. Schmid, $rDAN$: Toward robust demand-aware network designs, Inf. Process. Lett. 133 (2018).

[90] C. Avin, K. Mondal, S. Schmid, Demand-aware network designs of bounded degree, in: DISC, 2017.

[91] S. Schmid, C. Avin, C. Scheideler, M. Borokhovich, B. Haeupler, Z. Lotker, Splaynet: Towards locally self-adjusting networks, IEEE/ACM Trans. Netw. 24 (3) (2016) 1421–1433.

[92] C. Avin, K. Mondal, S. Schmid, Demand-aware network design with minimal congestion and route lengths, in: INFOCOM, 2019.

[93] K.-T. Foerster, S. Schmid, S. Vissicchio, Survey of consistent software-defined network updates, IEEE Commun. Surv. Tutor. 21 (2) (2019) 1435–1461.

[94] T. Fenz, K.-T. Foerster, S. Schmid, A. Villedieu, Efficient non-segregated routing for reconfigurable demand-aware networks, in: IFIP Networking, IEEE, 2019.