# Efficient Non-Segregated Routing for Reconfigurable Demand-Aware Networks

Thomas Fenz Klaus-Tycho Foerster Stefan Schmid Anaïs Villedieu
Faculty of Computer Science, University of Vienna, Austria

***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 mixed integer 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.**

## I. 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], [3], [4], [5], [6], [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], [12], [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

ISBN 978-3-903176-16-4 

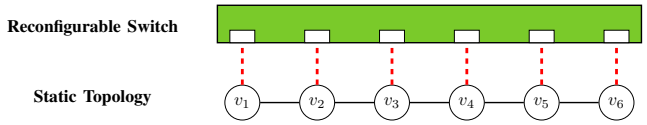



Fig. 1: In this small example, six nodes $v_1, \ldots, v_6$ are connected to a reconfigurable switch (connections dashed): the network operator can choose a matching of nodes inside the switch, creating a direct path between two nodes each time. Consider the case where there is a demand $v_1 \rightarrow v_5$ 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 $(v_6, v_5)$.

(see e.g., [9], [14]), or their algorithms rely on overly simple matching-cased heuristics (see, e.g., [8], [15], [16], [17], [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.

### A. 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 workload-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.
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 workload-driven simulations (based on Facebook 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.

### B. Organization

We describe our formal model in Section II and provide inapproximability NP-hardness results in Section III. We then discuss various routing algorithms in Section IV, ranging from a mixed integer program to greedy heuristics. The performance of these algorithms is then evaluated in Section V, where we use segregated routing as a baseline. Lastly, we discuss related works in Section VI and conclude in Section VII.

## II. 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 $V = \{v_1, \ldots, v_n\}$ (*e.g.,* top-of-the-rack switches), using 1) (usually electrical) 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 III, 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 $D$.

## III. 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 $v_1, \ldots, v_k$, 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 $i$th 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 $P = NP$.*

*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,

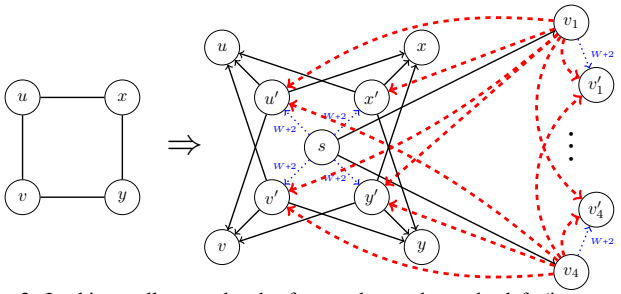



Fig. 2: In this small example, the four node graph on the left (instance $I$) is transformed to the $4 \cdot 4 + 1$ node graph on the right (instance $I'$). In $I'$, all unmarked links (static or reconfigurable ones, dashed) have a cost of 1; reconfigurable links with a weight of greater than $W+2$ are 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 $v'_1, \ldots, v'_4$, 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 $v_1, \ldots, v_4$ can only create one outgoing reconfigurable link. When matching to nodes from $v'_1, \ldots, v'_4$, 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 $v'_1, \ldots, v'_4$ 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.

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 $s$, $v_1, v_2, \ldots, v_{|V|}$, and $v'_1, v'_2, \ldots, v'_{|V|}$, connecting $s$ to each of the nodes $v_1, v_2, \ldots, v_{|V|}$ 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 Figure 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 can 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 $\Omega(\log n)$. ■

## IV. 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 IV-A, followed by an algorithm for a single flow in Section IV-B. We then provide a wide range of polynomial-time general heuristics, starting by prioritizing large demands in Section IV-C respectively large demands w.r.t. their initial routing distance in Section IV-D, followed by algorithms that greedily add paths (IV-E) or links (Section IV-F).

### A. An ILP for Demand-Aware Routing

We now present our integer linear program (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 $j$, $o^R_{ij}$ represents the weight of the reconfigurable link from $i$ to $j$ in switch $R$ 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 switch $R$ by setting the value of $m^R_{ij}$ to 1. The boolean $x^{st}_{ij}$ is set to 1 if a link from $i$ to $j$ is used in the shortest path from $s$ to $t$, as well as $y^{st}_{ij}$ if that link is a reconfigurable one. Finally, the length of the shortest path from $s$ to $t$ is given by $d_{st}$.

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

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

**Constraints:** A node connected to a 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).

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

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

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

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

**Path cost:** The length of the path from the sender to the reciever 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.

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

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

$$(\sum_{R} m_{ij}^{R}) + x_{ij}^{st} - 1 \leq y_{ij}^{st} \tag{7a}$$

$$y_{ij}^{st} \leq \sum_{R} m_{ij}^{R} \tag{7b}$$

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

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 V.

### B. `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 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].

### C. `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.

### D. `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` (see Algorithm 2). With this algorithm we compute, for every demand, the improvement of the matching (w.r.t. to the objective value) created by `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:*

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

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

The complexity of this algorithm is $O(d^2 n^2)$ 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.

### E. 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 as in GainDemand after every demand gets routed.

### F. 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.

## V. Evaluations

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 the real-world workloads based on Facebook's datacenter traffic data [24], [25], [26].

### A. Methodology

In order to generate workloads, we use the data of Facebook cluster C, which features the most dense demands. We map the corresponding demands to the leafs (*i.e.,* the servers) of a balanced tree 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 optical links are set to 1, while the static link weights vary between experiments.

In order to tune the density of the demand, we aggregate requests over a time window, either 10s or 100s. We also note that the Facebook data is sampled, at a rate of 30,000. Furthermore, to increase the density of the demand and hence render the experiments more interesting, we interpret the rack-to-rack demands as server-to-server demands, between the leaves. Intra-rack communication is hence ignored.

All presented results are averaged values over 10 different starting times.

For our experiments, trace data was stored in MongoDB and and simulations executed on 4 identical VMs each with 24 cores. We used Intel Xeon CPUs E5-2650 v4 at 2.20GHz X 24, with 16 GB RAM and running Ubuntu 18.04.1 LTS. Experiments are implemented in Python version 3.7.1 using gurobipy, networkx, and munkres libraries. To solve the ILP,

**Algorithm 1:** DemandFirst

**Input**: A weighted hybrid network $N$ and traffic demands $D$.

1) Sort the demand entries in $D$ by size.
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 objective function values 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.

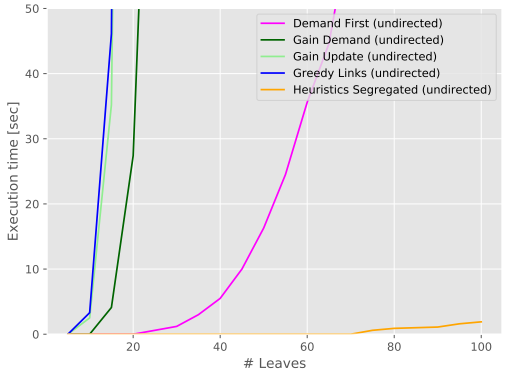
**Algorithm 4:** GreedyLinks

**Input**: A weighted hybrid network $N$ and traffic demands $D$.

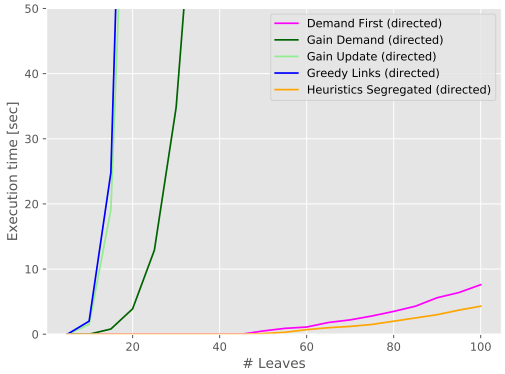
1) Initialize an empty list *value*.
2) Until no more matching links can be added to $N$:
   a) For every possible further reconfigurable link $e$ in $N$:
      i) Add $e$ as a matching link to $N$.
      ii) Compute the value of the objective function on $N$.
      iii) Add the value (for the link $e$) 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.




(a) Undirected, $n \leq 100$ servers, weight ratio: 1:5, time window: 100.




(b) Directed, $n \leq 100$ servers, weight ratio: 1:5, time window: 100.

Fig. 3: Comparison of execution time in seconds for the different heuristics. All results are averaged values over 10 different starting times from the Facebook data. We believe that the runtime of `DemandFirst` can be heavily improved in the undirected case, by employing optimizations for the considered weights.

we used Gurobi. In order to optimize the runtime of our ILP for the smaller instances, we used quadratic constraints rather than linear constraints, leveraging internal optimizations of Gurobi. The computed solutions remain unchanged.

We set a runtime limit of 20 minutes 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.

Notwithstanding, we provide an exemplary comparison in Fig. 3. Whereas the execution time of `DemandFirst` is similar to the segregated algorithm for the directed case, `DemandFirst` is significantly slower in the undirected case. The reason is that the subroutine `ReconfigDijkstra` in the undirected case is not optimized for the employed weights. Rather, it is a general approach, as it is not even clear if the problem is NP-hard for a single flow with arbitrary weights. We plan to improve the performance for the undirected case in future work, by employing a more specialized algorithm and a better utilization of optimized libraries such as networkx. We then expect the undirected case to behave similarly to the directed case, *i.e.,* roughly similar to the segregated matching algorithm for `DemandFirst`.

### B. 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[1] approaches for segregated routing in hybrid networks as a baseline, described next.

For the undirected case, we compute a maximum weight matching on the demand matrix, as done in *e.g., c-Through* [15]. Recall that only the leaf nodes act as sources and destinations in our setting. The unidirectional case is handled similarly, where we compute a maximum weight bipartite matching between the outgoing and incoming reconfigurable link ports, as suggested in *e.g., Helios* [16]. The routing is then performed in a segregated manner, where a route may either use a direct matching link or the static network parts. We denote these methods as *heuristics segregated* in the plots.

For the sake of completeness, we also include the routing performance without the optical circuit switch, denoted as *ilp static tree only* (we used the ILP for convenience reasons here).

[1]Standard with respect to the studied topology

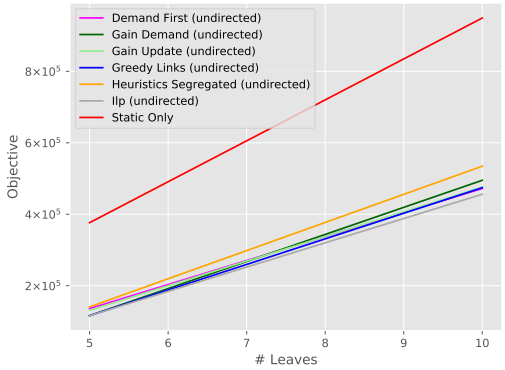
### C. Results

We first describe our results on very small networks, then expanding to networks to up to 100 nodes, all shown in Fig. 4.

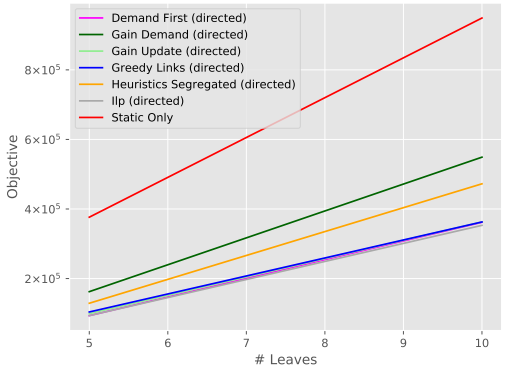
**Small examples.** Figures 4a and 4b provide a first impression of the optimization opportunities provided by the different algorithms (weight factor 1, time window 10), for a small network of up to $n = 10$ servers, due to the high runtime of the exact solution (the integer linear program). We observe that in these networks, while the optimal solution computed by the integer linear program (*ilp*) is always strictly better than all other algorithms, the difference is relatively small. But already here, we can see a price of segregated routing, which results in longer paths. 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.

**Larger networks.** For larger networks, the situation becomes more interesting, see Figure 4c for results on undirected networks (weight 5, time window 10): The heuristics `DemandFirst`, `GainDemand`, and `GainUpdate` perform consistently better than the segregated heuristic, however, `GainDemand` and `GainUpdate` are also much slower. Thus, we conclude that `DemandFirst` is the most attractive algorithm for such undirected networks.

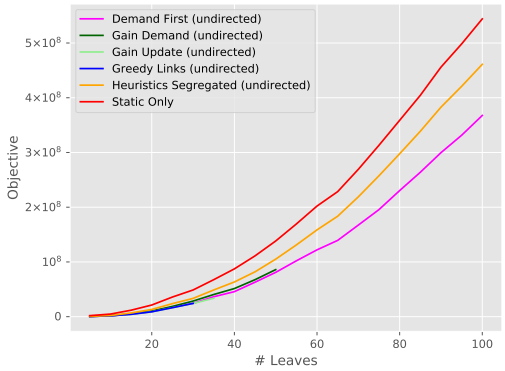
For directed networks networks, the situation is interesting as well: Figure 4d shows the results on directed scenarios (weight 5, time window 10): also here, `DemandFirst` provides the best results, also clearly outper-



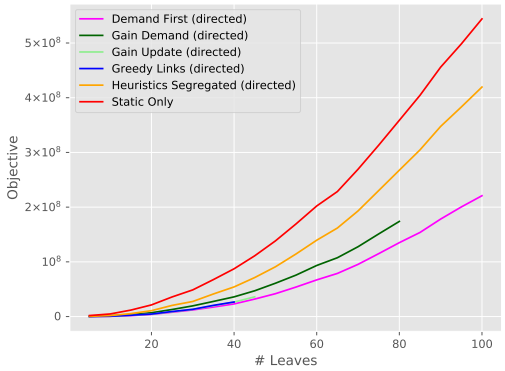

(a) Undirected, $n \leq 10$ servers, weight ratio: 1:1, time window: 10.



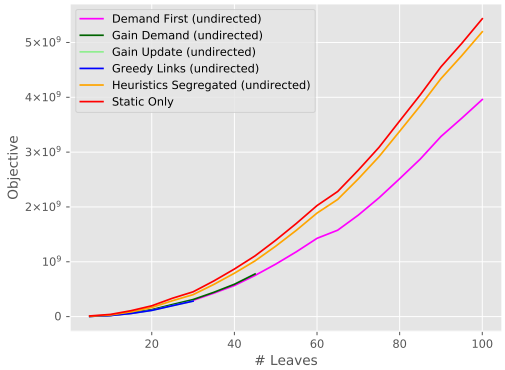

(b) Directed, $n \leq 10$ servers, weight ratio: 1:1, time window: 10.



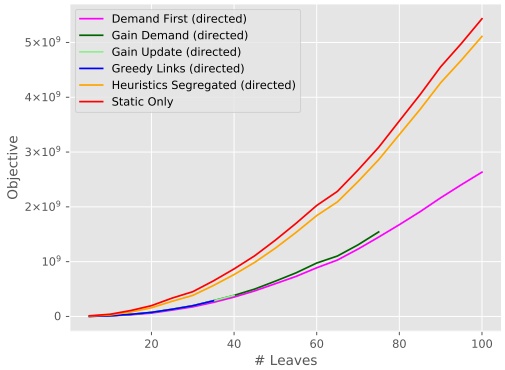

(c) Undirected, $n \leq 100$ servers, weight ratio: 1:5, time window: 10.



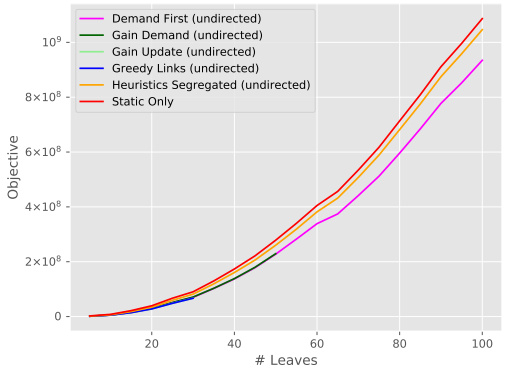

(d) Directed, $n \leq 100$ servers, weight ratio: 1:5, time window: 10.



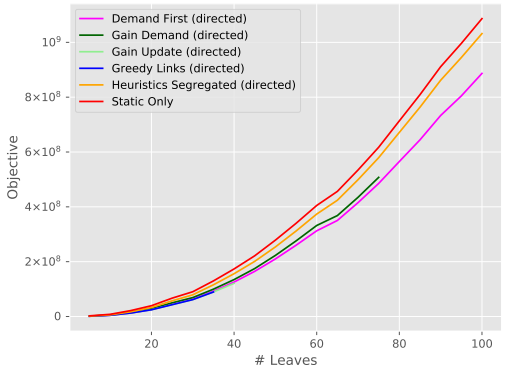

(e) Undirected, $n \leq 100$ servers, weight ratio: 1:5, time window: 100.




(f) Directed, $n \leq 100$ servers, weight ratio: 1:5, time window: 100.




(g) Undirected, $n \leq 100$ servers, weight ratio: 1:1, time window: 100.




(h) Directed, $n \leq 100$ servers, weight ratio: 1:1, time window: 100.

Fig. 4: Comparison of the achieved objective values (Eq. (1)) for the different heuristics and the segregated approach. Smaller values are better, as they represent the (weighted) average path lengths. The optimum value (ILP) is only computed for small networks, similarly, some slower heuristics do not finish their computation in the allotted time. Note that all results are averaged values over 10 different starting times from the Facebook data.

forming `GainDemand` and `GainUpdate` in terms of quality, while at the same time being much faster (lower runtime).

**Denser traffic.** Figures 4e and 4f show the results of the same experiments but where requests in the demand are aggregated over 100 units, resulting in denser demands. Here `DemandFirst` again performs best, very clearly outperforming the segregated algorithms. While the other of our heuristics come close, their computation time is too prohibitive currently for larger network instances, and might already run into issues at 30-40 nodes. Maybe interestingly, the performance of the segregated algorithms degrades even further, getting close to the static networks without the optical circuit switch.

**Identical weights.** In the prior experiments, we considered a 1-to-5 ratio for the weight of reconfigurable versus static links. However, when using technologies such as cheap converter switches [27], one can also consider the case where all link weights are created equal, though such networks usually have a much higher degree of topological reconfigurability. Figures 4g and 4h show the same experiments for the denser demands as before, but with a 1:1 weight ratio. As expected, the benefit of hybrid routing is much smaller, yet still clearly visible. Notwithstanding, `DemandFirst` again outperforms the segregated algorithms, with our other heuristics being close.

### D. Discussion

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 Facebook traffic traces. Interestingly, the simplest of our heuristics, `DemandFirst`, typically also provides the best results, making it an attractive solution in practice: its runtime is significantly lower than that of the other heuristics, and the provided route lengths shorter, both for undirected and directed scenarios. At the same time, we believe that `DemandFirst` can be refined further, and tailored to specific scenarios, which introduces an interesting avenue for future research. A first next step would be to improve the execution time of `DemandFirst` in the undirected case, we believe a similar performance as in the directed case is possible.

## VI. Related Work

The benefits and limitations of reconfigurable networks, which not only arise in datacenters but also in wide-area networks [28], [29], [30], [31], are currently discussed intensively in the literature, see e.g., [8], [9], [16], [28], [32], [33], [34], [35], [36]. 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 [27] to approaches based on rotor switches rotating through a set of pre-defined matchings [37]. 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* [38], [39] literature considering how to enhance a given (fixed) graph with an optimal number of "extra edges", sometimes also referred to as "ghost edges" [40]: the objective in this literature is typically to provide small world properties [41] or minimize the network diameter [42], [43]. However, most of these algorithms are not applicable directly to our model, where rather than individual edges, entire matchings can be added.

In this context, it could also be interesting to consider the removal of links, *e.g.,* for the scheduling of link repairs [44].

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 [15], [16], [17], [8], [18] and related concepts, such as edge-coloring [45] and stable-marriage schemes [9]. The closest paper to ours is the work by Foerster et al. [14], [46], 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 [47], [48], [49], [50], [51], [52].

## VII. 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 schemes, and to better understand which specific workloads and scenarios allow to improve quality and runtime further. Moreover, it would also be interesting to consider dynamic or online settings, possibly in conjunction with consistent network updates [53].

## Reproducibility

In order to simplify future research and in order to make our results reproducible, we created a code repository at https://gitlab.cs.univie.ac.at/ct-papers/2019-networking.


## Acknowledgments

We thank Andreas Blenk and Johannes Zerwas for several discussions that helped to improve this paper. We also thank the anonymous reviewers for their feedback.

## REFERENCES


[1] M. Noormohammadpour and C. S. Raghavendra, "Datacenter traffic control: Understanding techniques and tradeoffs," *IEEE Communications Surveys & Tutorials*, vol. 20, no. 2, pp. 1492–1525, 2017.

[2] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," in *SIGCOMM*. ACM, 2008.

[3] S. Kassing, A. Valadarsky, G. Shahaf, M. Schapira, and A. Singla, "Beyond fat-trees without antennae, mirrors, and disco-balls," in *SIGCOMM*. ACM, 2017, pp. 281–294.

[4] C. Guo, G. Lu, D. Li, H. Wu, X. Zhang, Y. Shi, C. Tian, Y. Zhang, and S. Lu, "Bcube: a high performance, server-centric network architecture for modular data centers," in *SIGCOMM*. ACM, 2009, pp. 63–74.

[5] A. Singla, C. Hong, L. Popa, and 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, and S. Sengupta, "VL2: a scalable and flexible data center network," in *SIGCOMM*. ACM, 2009, pp. 51–62.

[7] V. Liu, D. Halperin, A. Krishnamurthy, and 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, and 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, and D. C. Kilper, "Projector: Agile reconfigurable data center interconnect," in *SIGCOMM*. ACM, 2016.

[10] C. Avin and S. Schmid, "Toward demand-aware networking: a theory for self-adjusting networks," *ACM SIGCOMM Computer Communication Review (CCR)*, vol. 48, no. 5, pp. 31–40, 2018.

[11] M. Alizadeh, A. G. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan, "Data center TCP (DCTCP)," in *SIGCOMM*. ACM, 2010, pp. 63–74.

[12] A. Roy, H. Zeng, J. Bagga, G. Porter, and 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, and 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, and M. P. Ryan, "c-through: part-time optics in data centers," in *SIGCOMM*. ACM, 2010, pp. 327–338.

[16] N. Farrington, G. Porter, S. Radhakrishnan, H. H. Bazzaz, V. Subramanya, Y. Fainman, G. Papen, and 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, and G. Porter, "Circuit switching under the radar with reactor," in *NSDI*. USENIX, 2014, pp. 1–15.

[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, and A. C. Snoeren, "Scheduling techniques for hybrid circuit/packet networks," in *CoNEXT*. ACM, 2015, pp. 41:1–41:13.

[19] S. B. Venkatakrishnan, M. Alizadeh, and P. Viswanath, "Costly circuits, submodular schedules and approximate carathéodory theorems," in *SIGMETRICS*. ACM, 2016, pp. 75–88.

[20] U. Feige, "A threshold of ln $n$ for approximating set cover," *J. ACM*, vol. 45, no. 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," *Numerische Mathematik*, vol. 1, no. 1, pp. 269–271, Dec 1959.

[23] M. L. Fredman and R. E. Tarjan, "Fibonacci heaps and their uses in improved network optimization algorithms," *J. ACM*, vol. 34, no. 3, pp. 596–615, 1987.

[24] A. Roy, H. Zeng, J. Bagga, G. Porter, and A. C. Snoeren, "Inside the social network's (datacenter) network," in *SIGCOMM*. ACM, 2015.

[25] J. H. Zeng, "Data sharing on traffic pattern inside facebook's datacenter network," https://research.fb.com/data-sharing-on-traffic-pattern-inside-facebooks-datacenter-network/, Jan. 2017.

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

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

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

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

[30] R. Singh, M. Ghobadi, K.-T. Foerster, M. Filer, and P. Gill, "Radwan: Rate adaptive wide area network," in *SIGCOMM*. ACM, 2018.

[31] R. Singh, M. Ghobadi, K. Foerster, M. Filer, and P. Gill, "Run, walk, crawl: Towards dynamic link capacities," in *HotNets*. ACM, 2017.

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

[33] H. Liu, F. Lu, A. Forencich, R. Kapoor, M. Tewari, G. M. Voelker, G. Papen, A. C. Snoeren, and G. Porter, "Circuit switching under the radar with reactor." in *NSDI*. USENIX, 2014.

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

[35] X. Zhou, Z. Zhang, Y. Zhu, Y. Li, S. Kumar, A. Vahdat, B. Y. Zhao, and H. Zheng, "Mirror mirror on the ceiling: Flexible wireless links for data centers," *ACM SIGCOMM Computer Communication Review (CCR)*, vol. 42, no. 4, pp. 443–454, 2012.

[36] K. Chen, A. Singla, A. Singh, K. Ramachandran, L. Xu, Y. Zhang, X. Wen, and Y. Chen, "OSA: an optical switching architecture for data center networks with unprecedented flexibility," *IEEE/ACM Trans. Netw.*, vol. 22, no. 2, pp. 498–511, 2014.

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

[38] A. Gozzard, M. Ward, and A. Datta, "Converting a network into a small-world network: Fast algorithms for minimizing average path length through link addition," *Information Sciences*, vol. 422, 2018.

[39] A. Meyerson and B. Tagiku, "Minimizing average shortest path distances via shortcut edge addition," in *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*. Springer, 2009, pp. 272–285.

[40] M. Papagelis, F. Bonchi, and A. Gionis, "Suggesting ghost edges for a smaller world," in *Proc. 20th ACM International Conference on Information and Knowledge Management*, 2011, pp. 2305–2308.

[41] N. Parotsidis, E. Pitoura, and P. Tsaparas, "Selecting shortcuts for a smaller world," in *Proc. SIAM International Conference on Data Mining*. SIAM, 2015, pp. 28–36.

[42] D. Bilò, L. Gualà, and G. Proietti, "Improved approximability and non-approximability results for graph diameter decreasing problems," *Theoretical Computer Science*, vol. 417, pp. 12–22, 2012.

[43] E. D. Demaine and M. Zadimoghaddam, "Minimizing the diameter of a network using shortcut edges," in *SWAT*, 2010.

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

[45] L. Chen, K. Chen, Z. Zhu, M. Yu, G. Porter, C. Qiao, and S. Zhong, "Enabling wide-spread communications on optical fabric with megaswitch," in *NSDI*. USENIX, 2017, pp. 577–593.

[46] K.-T. Foerster, M. Pacut, and S. Schmid, "On the complexity of non-segregated routing in reconfigurable data center architectures," *ACM SIGCOMM Computer Communication Review (CCR)*, 2019.

[47] C. Avin and S. Schmid, "Toward demand-aware networking: A theory for self-adjusting networks," in *ACM SIGCOMM Computer Communication Review (CCR)*, 2018.

[48] C. Avin, A. Hercules, A. Loukas, and S. Schmid, "*rDAN*: Toward robust demand-aware network designs," *Inf. Process. Lett.*, vol. 133, 2018.

[49] C. Avin, K. Mondal, and S. Schmid, "Demand-aware network designs of bounded degree," in *DISC*, 2017.

[50] S. Schmid, C. Avin, C. Scheideler, M. Borokhovich, B. Haeupler, and Z. Lotker, "Splaynet: Towards locally self-adjusting networks," *IEEE/ACM Trans. Netw.*, vol. 24, no. 3, pp. 1421–1433, 2016.

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

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

[53] K.-T. Foerster, S. Schmid, and S. Vissicchio, "Survey of consistent software-defined network updates," *IEEE Commun. Surveys Tuts.*, 2018.