Distributed Construction of Demand-Aware Datacenter Networks

Aleksander Figiel[1] Darya Melnyk[1] Tijana Milentijević[1] Stefan Schmid[1,2]

[1]TU Berlin, Germany [2]Fraunhofer SIT, Germany

***Abstract*—Demand-aware reconfigurable datacenter networks adapt toward the traffic they serve by providing topological shortcuts between frequently communicating racks. However, only little is known about computing optimized demand-aware networks quickly and in a distributed manner. In this paper, we investigate fast distributed algorithms to compute demand-aware networks for hybrid datacenters, where a fixed capacitated network can be enhanced with a bounded-degree demand-aware network, i.e., with a set of matchings created by optical circuit switches.**

We make two main contributions. Firstly, we present a distributed algorithm, called the COORDINATOR algorithm for computing demand-aware networks on all underlying topologies. The algorithm is analyzed in the widely deployed Clos topology and in the Congested Clique model, where it is optimal in terms of quality and nearly optimal in distributed runtime.

Secondly, we focus on improving the round complexity at the cost of the quality of the resulting topology. We show that for tree demands, an adaptation of a distributed matching algorithm by Wattenhofer and Wattenhofer (DISC 2004) achieves a $1/6$-approximation. Based on this approach, we introduce the PROPOSE AND REJECT algorithm for general demands, which we evaluate on real-world Facebook datacenter and HPC traces. Our results show that the PROPOSE AND REJECT algorithm, even with limited knowledge of the demand matrix, performs nearly optimally on real traffic demands and covers over 80% of the demand. This is achieved with significantly fewer communication rounds than the optimal solution computed by the COORDINATOR algorithm.

***Index Terms*—distributed algorithms, demand-aware networks, reconfigurable datacenters, Clos topologies**

I. INTRODUCTION

Datacenters have become a critical infrastructure of our digital society, providing scalable, available, and reliable cloud computing services [1], [2]. Given the popularity of data-centric applications and artificial intelligence, however, the traffic in datacenters is growing explosively, imposing increasingly stringent requirements on the performance of the underlying networks.

Existing datacenter networks are designed to serve very general traffic patterns, and provide attractive properties such as full bisection bandwidth, bounded network diameter, and high expansion [3]. In other words, these networks are optimized toward all-to-all communication traffic, thus maximizing performance in the worst case [4]. Furthermore, the underlying topology is fixed and oblivious to the actual traffic demand.

This project has received funding from the European Research Council (ERC) under the Grant agreement No. 864228 (AdjustNet), as well as from the German Research Foundation (DFG), grant 470029389 (FlexNets).

An intriguing alternative datacenter network design are demand-aware networks, whose topology is optimized toward the specific workload they serve. Demand-aware networks are enabled by emerging reconfigurable optical communication technologies [5] and are empirically motivated by the fact that datacenter traffic features much spatial and temporal structure [6]–[8]. By accounting for the actual traffic demand, demand-aware networks can reconfigure themselves to provide topological shortcuts between frequently communicating endpoints. Accordingly, fewer network resources are consumed by such elephant flows, improving network capacity and minimizing "bandwidth tax" [9]–[13]. However, today we still do not have a good understanding of the fundamental algorithmic problem underlying such demand-aware network designs. In particular, existing literature on the demand-aware bounded-degree network design problem [8], [14]–[17] considers centralized algorithms assuming complete knowledge of the demand matrix. Such centralized algorithms may however not scale and introduce delays.

This paper initiates the study of distributed algorithms to compute demand-aware networks based on a b-matching model. In our model, nodes (e.g., top-of-rack switches) initially only have knowledge of their own demands towards other nodes, and with minimal information exchange, aim to establish a demand-aware network in a collaborative manner. In particular, we consider a typical hybrid datacenter network as is often studied in the literature [7], [11], [18], [19], where the demand-aware network is built on top of a fixed network. The demand-aware network is realized via optical circuit switches, where each such optical switch augments the topology with a constant number of matchings. We also assume that the communication demands are sparse. This assumption is justified by datacenter measurement studies [6], [7], [20], [21]. For example, Ghobadi et al. [7] showed that only 1% of rack pairs exchange traffic and up to 0.3% of rack pairs are accountable for more than 80% of the traffic. This property allows us to design efficient distributed communication algorithms.

A. Contributions

We initiate the study of distributed algorithms to enhance a given fixed network with a bounded-degree demand-aware network. In particular, we consider the Congest model [22] of distributed computing, where communication across the fixed

network is subject to capacity constraints and needs to be performed efficiently.

We first consider the basic problem of how to compute a demand-aware network in the congested clique model. We present a distributed algorithm that only requires a constant number of communication rounds to compute an optimal demand-aware network if the demand matrix is sparse. In particular, we propose a COORDINATOR algorithm that can solve the Demand-Aware Network (DAN) design problem optimally in 4 rounds, if the demand matrix is sparse and contains $n-1$ entries, where n is the size of the network. If the demand matrix contains $c \cdot n$ non-null entries, it can be solved optimally in $c+3$ rounds.

We next consider the Clos topology (the predominant datacenter network topology) and show that the COORDINATOR algorithm terminates in $4 \cdot (\sqrt[3]{\rho})^2 + 12$ rounds, if the demand matrix is sparse. Here, ρ denotes the number of roots of the multi-rooted FatTree that the Clos topology represents. This means that the number of communication rounds is in $o(n)$, where n is the number of racks at the leaves (i.e., leaves of the Clos topology).

In order to reduce the number of communication rounds, we show that the DAN design problem where the degree of the DAN is bounded corresponds to solving the b-matching problem in the graph. We then use distributed approximation algorithms to compute a b-matching in few rounds at a cost of an approximate solution. We start by showing that the Tree-Matching algorithm by Wattenhofer and Wattenhofer [23] for computing a $1/4$-approximation of an optimal matching where the demand matrix is a tree can be extended to compute a $1/6$-approximation of the optimal b-matching on trees. We then propose the PROPOSE AND REJECT algorithm that generalizes the Tree-Matching algorithm to work for general graphs. In the best case, this algorithm takes $O(b \cdot L)$ rounds to compute a DAN, where b is the highest degree in the DAN and L is the number of levels in the Clos topology.

To complement our analytical insights, we conduct experiments with the algorithms in the Clos topology. We evaluate our algorithms on Facebook, HPC and pFabric datasets. Our results show that the PROPOSE AND REJECT algorithm outperforms the COORDINATOR algorithm in terms of the number of rounds. In addition, the PROPOSE AND REJECT algorithm provides nearly optimal solutions for real traffic demands.

As a contribution to the research community, and in order to ensure reproducibility and facilitate follow-up work, we make all our implementations and experimental artifacts publicly available at https://github.com/inet-tub/dist-dan.

B. Organization

The remainder of this paper is organized as follows. We review related work in Section II and define our model in Section III, we propose the COORDINATOR algorithm and analyze it in the congested clique in Section IV. Further, we extend these results to the Clos topology. In Section V, we adapt the Tree-Matching algorithm to work for b-matchings on trees, and introduce the PROPOSE AND REJECT algorithm as a generalization of the Tree-Matching algorithm for general graphs. This algorithm is then evaluated and compared to the COORDINATOR algorithm on real-world data in Section VI. This paper concludes in Section VII.

II. RELATED WORK

Motivated by emerging optical technologies, reconfigurable datacenters and topology engineering have recently received much attention in the networking community [7], [9], [10], [13], [17], [24]–[38], see also the recent survey by Hall et al. [5]. Reconfigurable Datacenter Networks (RDCN) come in two flavors: *oblivious*, and *demand-aware* [3], [5]. Oblivious RDCNs such as RotorNet [9], Opera [10], Sirius [24], Mars [13], and Shale [39] rely on quickly and periodically changing interconnects between racks, to emulate a complete graph. Such emulation was shown to provide high throughput and is particularly well-suited for all-to-all traffic patterns [11]. In contrast, demand-aware RDCNs allow to *optimize* topological shortcuts, that depend on the traffic pattern. Demand-aware networks such as ProjecToR [7], SplayNets [28], Gemini [26], ReNet [19] Cerberus [11], or Duo [12] among others [33]–[38], [40], are attractive since a large fraction of communicated bytes belongs to a small number of elephant flows [6], [7], [34], [41]–[43]. By adjusting the datacenter topology to support such flows, e.g., by providing direct connectivity between intensively communicating source and destination racks, network throughput can be increased further (even if done infrequently [26]).

In this paper, we consider the fundamental problem of constructing a bounded degree DAN, which has already received much attention in the literature [8], [14]–[17]. In particular, it has been shown that there is an intriguing connection between the bounded-degree network design problem and the conditional entropy of the demand matrix, and an asymptotically optimal algorithm has been presented for sparse demand matrices [8]. However, all existing algorithms are centralized and assume complete knowledge of the demand matrix.

From the viewpoint of distributed algorithms, computing a demand-aware network in our model is equivalent to finding an optimal b-matching on the demand graph. Most work in this area has been conducted on optimal 1-matchings [23], [44]–[46]. For b-matchings, it has been shown that a $\frac{1}{6+\varepsilon}$-approximation can be achieved by a randomized algorithm in $O\left(\frac{\log^3 n}{\varepsilon} \log^2 \frac{P_{max}}{P_{min}}\right)$ rounds [47], where $\varepsilon > 0$ is a small constant, and P_{max} and P_{min} are the largest and the smallest edge weights respectively. Later, Fischer [48] presented a deterministic algorithm that can achieve a $\frac{1}{2+\varepsilon}$-approximation in $O\left(\log^2 \Delta \cdot \log \frac{1}{\varepsilon} + \log^* n\right)$ rounds.

Regarding distributed construction of demand-aware networks, we are only aware of one distributed approach in the literature. Avin et al. [49] remarked in their paper on square root graph sparsification that their algorithm can also be run in a distributed manner. However, their algorithms compute demand-aware networks whose degree depends on properties of the demand matrix, and can not be freely chosen to match

for example hardware constraints. Consequently, the resulting degree of the DAN might be too large to realize in practice.

In this paper, we contribute to the study of distributed algorithms for demand-aware network construction and present efficient solutions on how nodes can collaboratively compute optimal or near-optimal demand-aware networks across a fixed, capacitated network.

III. Preliminaries and Model

We assume that n nodes (e.g, top-of-rack switches in datacenters) from the set $V = \{1, 2, ..., n\}$ interact with each other following a sequence of communication requests σ, where $\sigma_i = (u, v) \in V \times V$ with source node u and destination node v. The requests in this sequence are drawn from a discrete distribution D (the demand) over all communication requests. The distribution D is represented by a demand matrix $M_D[p(i, j)]$ of size $n \times n$. An entry $p(i, j)$ corresponds to the probability that the source i communicates to the destination j. We further assume that the demand matrix is normalized and the diagonal entries $p(i, i)$ are set to 0.

In the static variant of the demand-aware network design problem, we are given a set of nodes V and the distribution D. The objective is to design a demand-aware network N_D in order to serve as many of the arriving communication requests through direct links as possible. We restrict the DAN to be selected from a family of desired topologies $\mathcal{N}$. In particular, we want the demand-aware network to have a bounded degree Δ, as switches have a bounded number of ports. Additionally, the demand-aware network includes all edges of the underlying topology. Consequently, we design an overlay network through reconfigurable optical networking switches to augment the existing fixed topology. Therefore, in our model, computing the overlay network for a specific demand matrix is equivalent to solving the weighted b-matching problem, where $b = \Delta - k$, where k is the degree of the underlying topology. In this b-matching problem, the goal is to match each node of the graph, represented by the demand matrix M_D, with at most b other nodes in a way that maximizes the total weight of the matched edges. Note that the overlay network can be recomputed at regular intervals, depending on the operational requirements.

In the distributed setting, nodes do not know the whole demand matrix. Instead, each node v knows how much traffic flows from v to all other nodes in the network. That is, each node has knowledge of its row in the matrix M_D. Further, the local computation power of the nodes is not bounded. The goal of the distributed DAN problem is to compute a DAN N_D in the network in a small number of rounds and notify the nodes of their new neighbors in N_D.

For the theoretical analysis of number of communication rounds, we consider two underlying topologies: clique, where the n nodes form a complete graph, and the Clos topology, which is a standard datacenter topology. Our setting for communication is the Congest model of distributed computing [22], where the message size for all messages is bounded by $O(\log n)$ bits. The communication model is synchronous, i.e. the nodes communicate in rounds.

IV. Distibuted Algorithms with Coordinator

In this section, we first consider distributed algorithms which however rely on a coordinator. For this model, we first consider a setting where the underlying topology is a clique and then Clos topology. Later, in Section V, we decentralize the algorithms in order to improve their round complexity.

A. Distributed DAN Computation in Congested Clique

In this section, we propose a similar, but simpler algorithm to Lenzen's routing algorithm [50] in order to compute the DAN in the congested clique model. This algorithm is based on a coordinator that computes the DAN for all nodes. In order to forward row entries to the coordinator fast, nodes in the network need help from their peers, as they might exceed the capacity restrictions otherwise.

In the first round, all nodes (except for the coordinator) broadcast how many entries their row contains. Based on the number of row entries, the nodes are divided into two subsets X and Y, where $X \cap Y = \emptyset$. If the matrix contains $c \cdot n$ entries, where c is a constant, then the subset X represents the set of nodes, where the number of row entries is at most $c + 1$. Subset Y is the set of nodes with at least $c + 2$ row entries. The subsets X and Y are referred to as "node helpers" and "nodes in need of help", respectively. In the second round, all nodes can send one value to the coordinator. In addition, the nodes in Y can send their values to the nodes in X in a predefined order. In further iterations, nodes in X forward the previously received values to the coordinator, whereas nodes in Y send their values to the coordinator and to their predefined helper nodes from X. In the penultimate round, the coordinator receives all values from the demand matrix and computes the demand-aware network either optimally in super-polynomial time or by using an approximation algorithm, e.g. [8]. When the network is computed, the coordinator sends a message back to each node with the information about which node pairs should construct edges in the last round.

Algorithm 1 presents the coordinator strategy as pseudocode.

Theorem 1. *The* Coordinator *algorithm solves the DAN design problem optimally in up to $c + 3$ communication rounds in the congested clique model, when the demand matrix contains up to $c \cdot (n + 1) + 2$ non-null entries for any $c \in \mathbb{N}$, where n is the number of nodes.*

Proof. In Algorithm 1, at least $2c + 2$ values can be communicated to the coordinator directly within the first two rounds. This is because each node can have at most $n - 1$ non-null values and therefore at least c nodes have at least two values. Two additional values can be added to any row, as any node (including the node helpers) can forward two values within the first two rounds. Note that node helpers do not forward any values in the first or in the second round.

Observe further that no values are sent to the coordinator in the last round. This round is used only by the coordinator to broadcast the final result to all nodes in the network.

Algorithm 1 COORDINATOR Algorithm for Demand-aware Network in Congested Clique Model

CO $\leftarrow$ an arbitrary node ▷ this is the coordinator
$x_i \leftarrow$ number of non-null entries in row i, for each i
for each node i **do**
  broadcast $x_i - (c+1)$ ▷ except the CO
  if $x_i \geq 1$ **then**
    send one entry from i to CO
  end if
end for
$s \leftarrow 1$; $t \leftarrow 1$; $X \leftarrow$ empty list
for each node i **do** ▷ except the CO
  if $x_i \leq c+1$ **then**
    X.append(i)
  else
    for each entry e in i'th row **do**
      $Y[t][e] \leftarrow i$
    end for
    $t \leftarrow t+1$ ▷ A node can have multiple entries, Y saves all entries
  end if
end for
$p \leftarrow 1$
for each t from $Y[t][e]$ **do**
  for each entry e **do**
    while $x_{X[p]} \geq c$ **do**
      take the next node from X
    end while
    $Y[t][e]$ sends one entry to X[p]
    $x_{X[p]} \leftarrow x_{X[p]} + 1$
    $p \leftarrow p+1$
  end for
end for
send one entry from i to CO if $x_i \leq 1$
each node from X sends its entry to CO

In the following, we will analyze how many values can be sent and forwarded in the rounds $3, \ldots, c+2$ to the coordinator. Note that at most $c(n-1)$ entries need to be sent to the coordinator in these rounds. We define the nodes that hold at least c values to be nodes in need of help (set Y), and the rest to be helpers (set X). Observe that, on average, each node holds at most c values. Therefore, there are always enough helpers and rounds to forward the values in Y. It remains to show that the helpers can receive all values that need to be forwarded without delay. Note that in Round 2, nodes in Y can send all values to the helpers that need to be forwarded in Round 3. The same holds for all following rounds up to round $c+1$. This is because all values in Y are ordered and can be sent in this order to the available helpers. If necessary, one node in Y can send values to several helpers in X in one round, since no other information is exchanged between the nodes after Round 1. This concludes the last part of the analysis. □

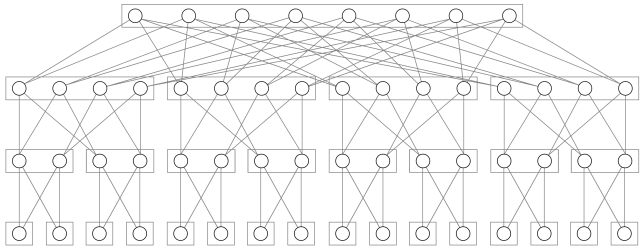


Fig. 1: Clos topology with levels 0, 1, 2, 3 (top to bottom) and degree k = 4

We would like to note that the presented lower bound on the number of exchanged messages is tight for the COORDINATOR algorithm in the case $c = 1$. For larger values of c, a constant number of additional messages can be handled by the algorithm within $c+3$ communication rounds.

B. Distributed DAN Computation in Clos Topology

Having studied how to compute a DAN in congested clique, in this section, we extend our investigations to study how to enhance the wide-spread Clos network with a demand-aware network. Clos networks are multi-rooted FatTrees and are the most widely deployed datacenter networks today, providing scalability and high performance [4]. A FatTree is a multi-rooted tree structure that contains multiple bipartite subgraphs. The topology is based on a three-tier design that holds a core level at the root, and an edge level at the leaves, where racks are placed, and an aggregation level in the middle. Let k be the highest degree of the graph and $L+1$ the number of levels. The number of nodes, i.e. roots in level $l = 0$ is $(\frac{k}{2})^L$. Each level l contains $2 \cdot (\frac{k}{2})^L$ nodes [4]. Nodes are divided into $(\frac{k}{2})^{L-l}$ blocks on each level l. The top level $l = 0$ contains only one block and all nodes are directly connected to one node from each of $(\frac{k}{2})^{L-1}$ blocks from level $l = 1$. Each node on level L is directly connected to $\frac{k}{2}$ nodes on level $L-1$, while each rack on level l, where $0 < l < L$ is directly connected to $\frac{k}{2}$ nodes from level $l+1$ and $\frac{k}{2}$ nodes from level $l-1$. In Figure 1, we visualize a Clos topology with 8 roots and 4 levels.

1) The COORDINATOR *Algorithm for Clos Topology.:* In the following, we adapt the COORDINATOR algorithm to the Clos topology. We assume that the topology contains 4 levels, numbered from 0 to 3. There are ρ roots and 2ρ servers, that are found at the last level $l = 3$. It is assumed that the demand matrix M_D contains $2\rho - 1$ entries and each server has knowledge only of its own row in the matrix. In order to compute the demand-aware network, all entries have to be sent to a coordinator that is chosen at the beginning of the algorithm. After receiving $2\rho-1$ entries, the coordinator has to compute the DAN and send a message to each server with the information on what edges should be formed in the DAN. The number of communication rounds needed for this algorithm is $4(\frac{k}{2})^{L-1} + 4L$, where k is the highest degree of the FatTree

and $L+1$ the number of levels. $2(\frac{k}{2})^{L-1}+2L$ rounds are needed to send the entries to the coordinator and $2(\frac{k}{2})^{L-1}+2L$ rounds for the coordinator to send back messages to all servers. The servers can then build edges accordingly. Observe that $\rho=(\frac{k}{2})^L$. The expression for the number of communication rounds for $L=3$ can be therefore rewritten as $4\cdot(\sqrt[3]{\rho})^2+12$.

Lemma 1. *Let ρ be the number of roots on level $l=0$. Then, the DAN design problem can be solved in $4\cdot(\sqrt[3]{\rho})^2+12$ communication rounds in the Clos topology where the demand-aware matrix contains $n-1$ entries.*

Proof. In order to compute the demand-aware network, all racks at level $l=3$ have to send their entries to an arbitrary coordinator in the graph. The coordinator computes the DAN and it replies to all servers at level $l=3$ which edges should be constructed. This process can be divided into two parts. The first part refers to all the racks sending their entries to the coordinator and lasts up to $2\cdot(\sqrt[3]{\rho})^2+6$ communication rounds. This will be proved by induction in the following. The second part requires $2\cdot(\sqrt[3]{\rho})^2+6$ communication rounds. The coordinator has to send to each rack from level $l=3$ one message (in total $2(\frac{k}{2})^L$ messages). Since the coordinator is an arbitrary rack from level $l=3$, it has degree $\frac{k}{2}$. Therefore, $2(\frac{k}{2})^L+2L$ or $2\cdot(\sqrt[3]{\rho})^2+6$ communication rounds are needed, where $L=3$ and $\rho=(\frac{k}{2})^L$.

Base case: Let $\rho=8$ be the number of roots in a FatTree, $n=16$ the number of racks on each level $0<l\leq 3$, and M_D an $n\times n$ demand-aware matrix with $n-1=15$ entries. Sending these entries from racks in the bottom level $l=3$ to the roots takes 3 rounds. The congestion occurs when sending the entries from roots to the coordinator, since the number of pods on each lower level increases. Sending entries from level 0 to level 1, and from level 1 to level 2 takes two communication rounds each. Sending entries from level 2 to the coordinator on level 3 takes four more rounds. This results in $11<2(\sqrt[3]{8})^2+6$ rounds.

Hypothesis: Let M_D be an $n\times n$ demand-aware matrix. $n-1$ entries can be sent to a coordinator with n racks with $x=8t, t\in\mathbb{N}$ roots in $2\cdot(\sqrt[3]{\rho})^2+6$ communication rounds.

Induction step: We prove that $2\cdot(\sqrt[3]{\rho+8})^2+6$ rounds are needed in in the Clos topology with $\rho+8$ roots and $n=2\rho+16$ servers on level $l=3$ in order to transmit $n-1$ entries to the coordinator. According to the hypothesis, 2ρ entries can be sent over ρ roots in $2\cdot(\sqrt[3]{\rho})^2+6$ rounds. The remaining 8 roots have to forward 15 messages. This matches the base case and can be done in 11 rounds. Observe, that some communication rounds can be spared when sending messages from servers at level $l=3$ to the roots and from the roots to racks at level $l=1$. It is possible to send the 2ρ entries from the hypothesis and 15 entries from the base concurrently. This lowers the number of rounds by 5. The total number needed for this transmission is $2\cdot(\sqrt[3]{\rho})^2+12$. Observe further that the inequality $2\cdot(\sqrt[3]{\rho})^2+12\leq 2\cdot(\sqrt[3]{\rho+8})^2+6$ holds for all $\rho\in\mathbb{N}$. Therefore, sending $n-1$ entries in a topology with $\rho+8$ roots takes at most $2\cdot(\sqrt[3]{\rho+8})^2+6$ rounds. □

Corollary 1. *Let the number of levels in the FatTree be $L=3$. If the number of communication rounds is noted as $f(k)$ and the number of servers as $g(k)$, where k is the highest degree of the FatTree, then $f\in o(g)$.*

V. Distributed Algorithms without Coordinator

In this section, we explore whether DANs can also be computed without coordinators. We start by presenting a distributed algorithm to compute a DAN for tree demands. We then use this idea to generalize the algorithm to general demand graphs. In order to compute a DAN from M_D, we transform the demand matrix into a corresponding graph, where an edge $\{i,j\}$ exists iff $p(i,j)>0$ and has weight $p(i,j)$. Then, we must choose a subset of edges with maximum weight, such that every vertex is incident to at most b edges. This problem is known as weighted b-matching.

A. Efficient Algorithms For Tree Demand

In this section, we show that it is possible to compute an almost optimal DAN without relying on a coordinator in the network. In particular, we present a distributed algorithm to compute a DAN where the demand matrix is a tree. Note that in Clos topology (see Figure 1), nodes are already connected with $\frac{k}{2}$ links. Since the degree of a DAN is bounded by a constant Δ, computing an overlaying DAN for a particular demand matrix corresponds to solving the weighted b-matching problem for $b=\Delta-\frac{k}{2}$ on a graph described by the matrix M_D. In the b-matching, the goal is to match each node of a graph with at most b other nodes such that the sum of the weights of all matched edges is maximized. Observe that many distributed algorithms solving the b-matching problem require a logarithmic number of rounds (either in the number of nodes or in the maximum degree of M_D). There are however also algorithms that only need a constant number of communication rounds on special graph classes. Here, we present an extension of an existing 1-matching algorithm on trees and show that this algorithm reaches a $1/6$-approximation of the optimal solution.

Our algorithm is based on the distributed matching algorithm for 1-matchings presented by Wattenhofer and Wattenhofer [23]. In this algorithm, each rack tries to establish a link to a neighboring rack with the highest demand. From all requests to establish a link, each rack picks the one with the highest demand. This way, the demand tree is split into disjoint paths. In order to compute a 1-matching, the authors further compute a maximal matching along each path. We adjust this algorithm to compute a 2-matching in the first step by letting the racks accept all disjoint paths as matching edges. For $b>2$, we repeat the algorithm on a graph, where the matched edges from the previous round are removed. Algorithm 2 presents this adapted version of the algorithm for any even $b>1$.

In the following, we will show that one round (inside the while loop) of the algorithm provides a $1/2$-approximation of the optimal 2-matching.

Lemma 2. *For $b=2$, Algorithm 2 achieves a $1/2$-approximation of the maximum weighted 2-matching.*

Algorithm 2 TREE Algorithm for Demand-aware Network in Clos Topology

while $b \geq 2$ **do**
  for each node u **do**
    u requests its heaviest incident edge e_u
  end for
  for each node u **do**
    u confirms the heaviest received request from e_v where $e_v \neq e_u$
    if u got a request from v **then**
      confirm e_u
    end if
  end for ▷ Each rack has at most degree 2. The resulting graph is a set of disjoint paths.
  delete confirmed edges
  $b = b - 2$
end while
All confirmed edges form a b-matching.

Proof. Let $\mathcal{A}$ denote Algorithm 2 and $w(\mathcal{A})$ denote the total weight of edges that were added in the first round of $\mathcal{A}$ to the b-matching. Wattenhofer and Wattenhofer [23] proved that the collection of paths and cycles computed in one round of $\mathcal{A}$ contains edges that have at least the same weight as the edges in a maximum 1-matching. Note that these edges can be mapped one to one to the edges in a maximum 1-matching. Let now M_1 be an optimal 1-matching with weight $w(M_1)$. A 2-matching M_2 in trees consists of two 1-matchings with weight $w(M_2) \leq 2 \cdot w(M_1)$. Therefore, the following inequality holds: $w(M_2) \leq 2 \cdot w(\mathcal{A})$. □

In order to show that the algorithm provides a $1/6$-approximation for any b, we will first compare the weight of the collection of b iteratively computed optimal 1-matchings to the weight of the optimal b-matching. We assume that in the iterative computation, the matched edges are deleted from the graph, and the new optimal matching is computed on the remaining graph. We refer to this strategy as b-iterative-1-matching.

Lemma 3. *The b-iterative-1-matching is a $1/3$-approximation of the b-matching.*

Proof. We start by considering matching edges that are present in the b-matching, but not in the b-iterative-1-matching. For b iterations, each such edge has not been added to any of the optimal 1-matchings. For the following analysis, we fix an edge e_b that is in the optimal, but not in the b-iterative-1-matching. Since the optimal 1-matching is maximal, e_b must have at least one neighboring edge, with which it shares a vertex, that is included in the optimal matching for each iteration. We can now use the assumption that we want to reach a $1/3$-approximation of the optimal b-matching, and assign to each edge of the iterative matching tokens that have three times the weight of this edge. In the following, we will redistribute the tokens among all edges that are part of the optimal b-matching as well as the b-iterative-1-matching, thus showing that the iterative algorithm is in fact a $1/3$-approximation of the optimal b-matching.

Consider the edge e_b in some iteration i. Assume that it has two neighboring edges $e_1(1)$ and $e_1(2)$ that are part of the optimal 1-matching in this iteration. Note that $w(e_b) < w(e_1(1)) + w(e_1(2))$ must hold since, otherwise, the optimal matching would have added edge e_b. We can now redistribute $\frac{1}{2b} \cdot 2 \cdot w(e_1(1))$ tokens from $e_1(1)$ to e_b, and $\frac{1}{2b} \cdot 2 \cdot w(e_1(2))$ tokens from $e_1(2)$ to e_b. Observe that this covers a $\frac{1}{b}$ fraction of the weight of e_b. If e_b has only one neighboring edge e_1 in the optimal 1-matching, then $w(e_b) < w(e_1)$. We can redistribute $\frac{1}{2b} \cdot 2 \cdot w(e_1)$ tokens to e_b, and thus cover $\frac{1}{b}$ of its weight.

Over b iterations, we can thus cover the whole weight of each edge e_b of the maximum b-matching. Next, we need to show that the edges of the optimal matching do not give up too many of their tokens, i.e. the remaining tokens should cover the total weight. This is true because each edge in the optimal matching can prevent at most $2b - 2$ neighboring edges from joining the b-matching. Thus, at least one third of the tokens will remain for each edge of the optimal matching. □

Using the previous results, we can now prove that Algorithm 2 indeed is a $1/6$-approximation:

Theorem 2. *Algorithm 2 computes a $\frac{1}{6}$-approximation of the maximum b-matching in trees.*

Proof. In this proof, we compare the solution of the algorithm to the b-iterative-1-matching from Lemma 3. We will show that, in each iteration, Algorithm 2 computes at least a $1/2$-approximation of the iterative solution. Together with Lemma 3, this will give us the $1/6$-approximation of the optimal result.

In Lemma 2, we discussed that the collection of disjoint paths resulting from the first step of the algorithm already contains edges that have a larger weight than an optimal 1-matching. Observe that an optimal 1-matching provides a $1/2$-approximation to the optimal 2-matching. Further, in each round of the algorithm, we compute a 2-matching on the edges that have not been chosen for a 2-matching in previous rounds. Thus, in each round, we compute a $1/2$-approximation of the optimal 2-matching on the remaining edges.

Observe that iteratively computed optimal 2-matchings ($b/2$-iterative-2-matching) provide at least a $1/3$-approximation of the optimal solution. This follows from Lemma 3 and the fact that a $2i$-iterative-1-matching has at most the same weight as any i-iterative-2-matching.

Algorithm 2 provides a $1/2$-approximation of the optimal 2-matching in each round after matched edges of the algorithm have been removed. This solution is also at least a $1/2$-approximation of the optimal iterative solution in the corresponding iteration, as the algorithm might have not matched edges from the optimal solution with better weight in previous rounds. Altogether, Algorithm 2 gives a $1/6$-approximation to the optimal b-matching for even b. □

In the following, we turn our attention to the number of communication rounds of the algorithm. The communication in Algorithm 2 takes place in the following parts: when each rack requests one edge and when each rack confirms up to two edges. In the best case, each rack requests a different rack, which takes $2L$ rounds, and each rack that has received a request sends up to two confirmed request messages in $2L$ rounds. This is repeated $\frac{b}{2}$ times. The worst case corresponds to the COORDINATOR algorithm (Algorithm 1). The number of rounds for requesting the same rack in all iterations is $\frac{b}{2} \cdot (2(\frac{k}{2})^{L-1} + 2L)$, and for confirming up to two edges is $\frac{b}{2} \cdot 4L$. This results in the following observation:

Observation 1. *The number of communication rounds needed in Algorithm 2 is $\frac{b}{2} \cdot 4L$ in the best case and $\frac{b}{2} \cdot (2(\frac{k}{2})^{L-1} + 2L) + \frac{b}{2} \cdot 4L$ in the worst case, where k is the highest degree of the graph and $L+1$ the number of levels in Clos topology.*

B. Distributed Propose-and-Reject Algorithm

In this section, we will introduce the PROPOSE AND REJECT algorithm for computing the DAN based on a demand matrix M_D. This algorithm is inspired by the Gale-Shapley algorithm for the stable matching problem [51] and is a simple generalization of Algorithm 2. Observe that also Wattenhofer and Wattenhofer [23] proposed an algorithm for approximating an optimal matching on general graphs. Their solution however does not require the resulting chosen edges to form a matching. The idea of our generalization is the following: every rack with non-null values starts by proposing to the rack to which it has the highest traffic. Racks can accept at most b proposals and they break ties depending on the highest traffic load. The DAN computed in such a way is not necessarily optimal, but the advantage is that the algorithm only needs few rounds to terminate.

In Algorithm 3, we present this idea in pseudocode. This algorithm consists of four different routines: Firstly, in Routine 1, each server orders its own entries from the matrix in descending order. After that, in Routine 2, all servers send a proposal containing the probability from the first entry to the corresponding server. The status of these servers is set to tentative, since they are waiting for one possible acceptance of the proposal. In Routine 3, all received proposals are examined, and are either accepted or rejected. The proposals are in the form (u, v, p) where u is the source server, v is the destination and p is the probability (frequency) of having traffic between u and v. Each server that has received a proposal has to add up the probability p with its own probability of the entry (v, u). These values are considered together with $v's$ non-used entries. The list of these values is ordered in descending order and only the first b_v entries of the list are taken into account. Proposals are accepted in accordance with satisfying the constraint that a rack v can have at most b_v edges. If v's state is tentative, then v is waiting for a possible match from another rack. In this case, there has to be one edge saved for this possible match and up to $b_v - 1$ edges can be formed. Racks that have accepted a proposal send a message to the source servers, in order for them to change their state to available (not tentative) in Routine 4. If servers do not receive any acceptance of the proposal or are rejected, their state must be switched to available for the next round of the algorithm starting at Routine 2. There are b iterations at most, since each rack can have up to b edges in DAN. All rounds except for the first round start at Routine 2, continue with Routine 3, and end with Routine 4.

Algorithm 3 PROPOSE AND REJECT

Routine 1: Sort the entries

1: **for** each server v **do**
2: order non-null entries in descending order
3: $b_v \leftarrow b$
4: **end for**

Routine 2: Send proposals

1: **for** each server u that has ordered non-null entries in the form $(u, v) \wedge b_u > 0$ **do**
2: send the first not used entry to v
3: $(u, v) \leftarrow used$
4: $u \leftarrow tentative$
5: **end for**

Routine 3: Receive and examine proposals

1: **for** each proposal (u, v, p) at rack v **do**
2: add p + probability in entry (v, u) to values[] ▷ v's preferences must also be considered
3: **end for**
4: add non-used entries of v to values
5: sort values[] in a descending order
6: consider the first b_v entries of values
7: **if** $v == tentative$ **then** ▷ did v send a proposal to another rack?
8: accept up to $b_v - 1$ proposals from values
9: **else**
10: accept up to b_v proposals from values
11: **end if**

Routine 4: Receive match

1: **for** each server v that receives accepted proposal **do**
2: $b_v \leftarrow b_v - 1$
3: $v \leftarrow available$ ▷ not tentative
4: **end for**
5: **for** each server v where $v == tentative$ **do** ▷ proposal is rejected
6: $v \leftarrow available$ ▷ cannot stay tentative
7: **end for**

1) Discussion of the Propose-and-Reject Algorithm.: In order to analyze the number of communication rounds of the modified version of the PROPOSE AND REJECT algorithm, the best and the worst cases are considered. The best case occurs when no two servers propose the same server. This implies that sending the proposals within an iteration can be represented by an injective function. Correspondingly, in the worst case,

all servers in each iteration send proposals to the same server. It is important to note that this algorithm is not limited to sparse matrices and works on all underlying topologies.

From Observation 1 we can derive the number of rounds that the PROPOSE AND REJECT algorithm takes in the Clos topology. The number of communication rounds in the best case with a sparse matrix is $4L$, where $L+1$ is the number of levels. In this case, each rack sends exactly one proposal, which needs $2L$ rounds, and each rack that has received a proposal sends an accepted proposal message in $2L$ rounds. In the best case with a non-sparse matrix, the number of rounds is $b \cdot 4L$, where b is the highest degree in the DAN. The explanation follows from the previous case: a non-sparse matrix represents b iterations of an algorithm for the sparse matrix, where each rack holds one entry in each iteration, and therefore the number of rounds is $b \cdot 4L$. The worst case in a sparse matrix corresponds to the COORDINATOR algorithm. The number of communication rounds is $2(\frac{k}{2})^{L-1} + \frac{b}{\frac{k}{2}} + 4L$, where k is the highest degree of the FatTree. Sending proposals to one rack takes $2(\frac{k}{2})^{L-1}+2L$ rounds. The rack has to accept b proposals and send them through its $\frac{k}{2}$ edges in the FatTree. In the worst case with a non-sparse matrix, the number of rounds is $b \cdot (2(\frac{k}{2})^{L-1} + 2L) + b \cdot (\frac{b}{\frac{k}{2}} + 2L)$. This can be derived from the previous case for a sparse matrix. Since the sparse matrix indicates one iteration of the algorithm, a non-sparse matrix contains up to b iterations. Hence, the number of communication rounds has to be multiplied by b.

VI. EXPERIMENTAL EVALUATION

In this section, we complement our analytical results and empirically evaluate the PROPOSE AND REJECT algorithm on different datasets and compare it to the COORDINATOR algorithm. We evaluate our PROPOSE AND REJECT and the COORDINATOR algorithm on real-world datacenter traffic [6] and analyze round complexity on a Clos topology.

A. Methodology

Datasets. For the practical evaluation, we used Facebook, High Performance Computing (HPC) and pFabric datacenter traces [6]. The Facebook dataset consists of three clusters: cluster A, cluster B and cluster C. Clusters A and C have 300M requests, whereas cluster B has 2B requests over 24 hours. Each trace entry contains a timestamp, a source, a destination rack, and some additional information such as packet length and IP protocol. For the evaluation, we consider 15 snapshots of 10 minutes, where each snapshot consists of 5900 to 15,000 racks with up to 15M requests. Since clusters A, B and C have dense traffic, we additionally extracted 15 snapshots of 10 minutes with sparse traffic from cluster C. The HPC data consists of 1024 racks and up to 20M requests, where each entry contains a sequence number, source and destination rack. The pFabric dataset contains 30M requests and 144 racks. For the evaluation, the HPC and pFabric traces were tested in their entirety, without being divided into smaller snapshots.

Due to the distributed nature of the presented algorithms, and in order to measure their round complexity in practice, the experiments were simulated on a widely adopted Clos topology with 3 levels. Note that adding more layers to the Clos topology will not change the quality of the solution. The number of communication rounds for both COORDINATOR and PROPOSE AND REJECT algorithm will increase, at most two rounds per layer.

Optimal solution. In order to compute the optimal solution for the COORDINATOR algorithm to which we will compare the PROPOSE AND REJECT algorithm, we solve the following ILP using Gurobi 11.0 [52]:

$$\begin{aligned} \text{maximize:} \quad & \sum_{e \in E} w(e) x_e \\ \text{subject to:} \quad & \sum_{u \in N(v)} x_{\{u,v\}} \leq b \qquad \text{for all } v \in V \\ & x_e \in \{0,1\} \qquad \text{for all } e \in E \end{aligned}$$

Here, the set E denotes the communication edges from the matrix M_D, and $w(e)$ the corresponding edge weights.

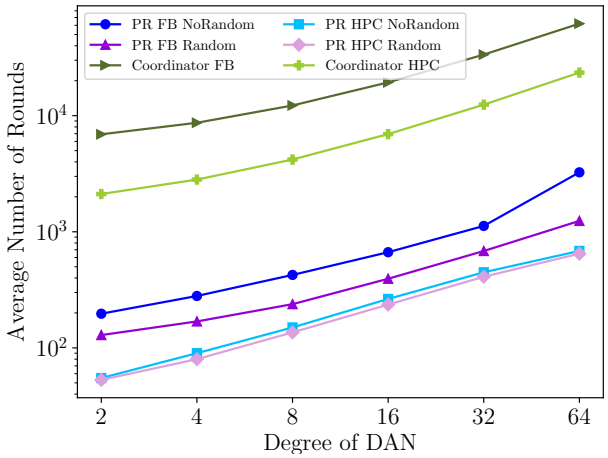
Randomized Propose-and-Reject algorithm. To overcome the problem of proposing to the same rack, we can randomize the transmission of proposals. The randomized PROPOSE AND REJECT algorithm changes Routine 2 in Algorithm 3 in the following manner: the first b proposals are sent in random order. This way, we reduce the probability of the racks to propose to the same rack and avoid congestion.

B. Results

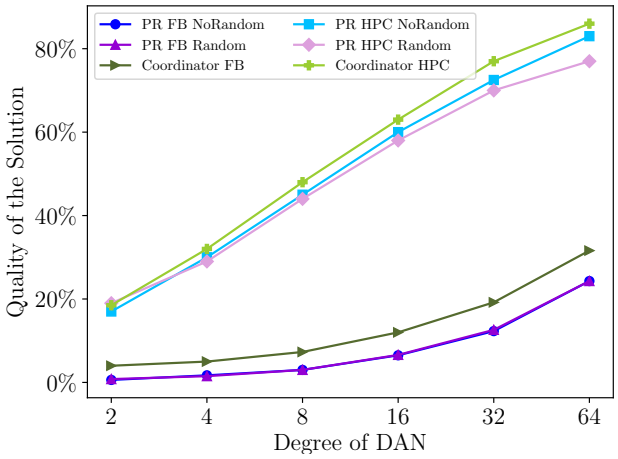
For the evaluation, we grouped the data into two sets: FB and HPC, and report on the averages in these sets. FB consists of 15 snapshots from all three clusters A, B and C, whereas HPC is comprised of HPC traces, pFabric traces and 15 snapshots of sparse cluster C. In this grouping the FB matrices are much denser than the HPC ones.

In Figure 2a, we compare the randomized and non-randomized PROPOSE AND REJECT algorithm to the COORDINATOR algorithm for the Facebook and HPC traces. We plot the average number of rounds for varying values for b. For the sake of comparison, we lower bound the number of entries that each rack can send to at most b messages in the COORDINATOR algorithm instead of communicating all the entries to the Coordinator. The communicated b messages represent the largest b entries of each rack. However, we assume that at the end of the communication phase the Coordinator has complete knowledge of all entries of the demand matrix.

Firstly, it can be concluded, that the COORDINATOR algorithm needs significantly more communication rounds than the randomized and non-randomized PROPOSE AND REJECT algorithm for both datasets. As previously explained, all source racks send b many messages to the Coordinator in the COORDINATOR algorithm, and the Coordinator sends back one message to all destination racks. Since the assumed Clos topology has 3 levels, the Coordinator can only send out 2 messages per round. Therefore, the number of rounds for the



(a) Number of rounds of the PROPOSE AND REJECT and COORDINATOR algorithm



(b) Solution quality of the PROPOSE AND REJECT and COORDINATOR algorithms.

Fig. 2: PROPOSE AND REJECT algorithm compared to the COORDINATOR algorithm in terms of the number of rounds and solution quality for varying maximum allowed degree b.

COORDINATOR algorithm grows proportionally to the number of destination racks. The PROPOSE AND REJECT algorithm, on the other hand, needs considerably fewer rounds than in the analytical worst-case computed in Section V-B. This suggests that there is no distinguished node in the demand matrix of the HPC data to which all other nodes are proposing. Further, we can see that the randomized PROPOSE AND REJECT algorithm needs fewer rounds than the non-randomized PROPOSE AND REJECT for the Facebook data. This is also illustrated in Figure 3. Facebook clusters encompass between 2M and 15M requests distributed across 200,000 to 600,000 entries in the demand matrix. These clusters exhibit a star-like structure and are regarded as dense. Randomization in this case solves the problem of proposing the same node and therefore avoids

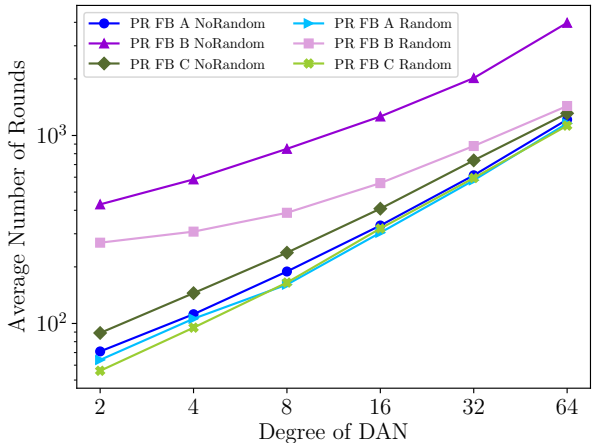


Fig. 3: Number of rounds of the randomized and non-randomized PROPOSE AND REJECT algorithm for Facebook clusters

congestion. The largest reduction in the number of rounds due to randomization is observed in cluster B, which is also the densest cluster, containing 15M requests and 600,000 entries in the demand matrix.

Figure 2b represents the achieved quality of the b-matching for the PROPOSE AND REJECT and the COORDINATOR algorithm. The quality of the solution measures how much of the communication is covered by the DAN. A solution of 100% quality means all edges of the graph are taken. Since the maximum degree b is bounded, it may not be possible to take all edges, however, the goal is then to take as many high weight edges as possible without breaking the degree constraint. The COORDINATOR algorithm delivers the optimal solution for each b, which is computed via an ILP solver in Section VI-A. Firstly, the HPC data surpasses the Facebook data regarding quality. The reason for this is that the HPC data is sparse, containing up to 10,000 entries in the demand matrix. In contrast, the Facebook dataset is dense, with up to 600,000 entries. Consequently, the COORDINATOR algorithm can cover no more than 35% of the graph for Facebook clusters. When comparing PROPOSE AND REJECT and the COORDINATOR algorithm for the Facebook data, it can be concluded that PROPOSE AND REJECT covers over 77% of the optimal solution computed by the COORDINATOR algorithm for $b = 64$. It is important to mention that the randomized and non-randomized PROPOSE AND REJECT deliver a DAN of same quality. Thus, for dense traffic it is beneficial to use the randomized PROPOSE AND REJECT, since the number of communication rounds is lower. For this see Figure 3. For the HPC data, the quality of the solution is at around 80% for the PROPOSE AND REJECT and 85% for the COORDINATOR algorithm. The HPC data is sparse and the average degree of its demand graph is 72. This results in higher graph coverage and better quality compared to the Facebook data. For larger values of b, randomization of the PROPOSE AND

REJECT algorithm is not very advantageous. For $b = 64$ the randomized PROPOSE AND REJECT covers 77% of the graph, whereas the non-randomized overlays 83%. For the sparse datasets, it is effective to use the non-randomized PROPOSE AND REJECT, since the number of rounds is smaller than in the COORDINATOR algorithm, while the quality remains unchanged (see Figure 2b).

VII. CONCLUSION

This paper initiated the study of distributed algorithms to efficiently compute a bounded-degree DAN across a fixed network. We presented three algorithms that are designed to compute a DAN in the congested clique model as well as to enhance general topologies, such as the Clos topology. Our results consist of a constant-time distributed algorithm in the Congest model, and two efficient algorithms when the fixed network is a Clos topology. We complemented our theoretical results with simulations and found that our PROPOSE AND REJECT algorithm, even with limited knowledge of the demand matrix, performs almost optimally on real traffic demands using significantly fewer communication rounds than the COORDINATOR algorithm's optimal solution. Note that our algorithms can be used to recompute the demand-aware networks periodically, with intervals tailored to the system's needs.

This paper leaves several interesting open questions for future work.

Our PROPOSE AND REJECT algorithm performs well on the Facebook dataset, benefiting from its sparse and clustered structure, which is characteristic for datacenter networks. It would be valuable to investigate whether similar patterns emerge in other datacenter traces and how more frequent reconfiguration could enhance result quality. Furthermore, analyzing the performance of both the PROPOSE AND REJECT and COORDINATOR algorithms in such scenarios would provide deeper insights. Sadly, the availability of datasets in this context is quite limited.

It would also be interesting to explore online algorithms for computing DANs dynamically, as well as generalizing the Clos topology to AB-FatTrees [53], which provide better fault tolerance than FatTrees.

REFERENCES

[1] A. Singh, J. Ong, A. Agarwal, G. Anderson, A. Armistead, R. Bannon, S. Boving, G. Desai, B. Felderman, P. Germano *et al.*, "Jupiter rising: A decade of clos topologies and centralized control in Google's datacenter network," *Proc. ACM SIGCOMM Computer Communication Review (CCR)*, vol. 45, no. 4, pp. 183–197, 2015.

[2] R. W. Alaskar and I. Ahmad, "Data center architectures: Challenges and opportunities," *International journal of new computer architectures and their applications*, pp. 117–129, 2014.

[3] C. Avin and S. Schmid, "Toward demand-aware networking: A theory for self-adjusting networks," *ACM SIGCOMM Comput. Commun. Rev. (CCR)*, vol. 48, no. 5, pp. 31–40, 2019.

[4] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," *ACM SIGCOMM Computer Communication Review (CCR)*, vol. 38, no. 4, pp. 63–74, Aug. 2008.

[5] M. N. Hall, K.-T. Foerster, S. Schmid, and R. Durairajan, "A survey of reconfigurable optical networks," *Optical Switching and Networking*, vol. 41, p. 100621, 2021.

[6] C. Avin, M. Ghobadi, C. Griner, and S. Schmid, "On the complexity of traffic traces and implications," *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, vol. 4, no. 1, pp. 1–29, 2020.

[7] M. Ghobadi, R. Mahajan, A. Phanishayee, N. Devanur, J. Kulkarni, G. Ranade, P.-A. Blanche, H. Rastegarfar, M. Glick, and D. Kilper, "Projector: Agile reconfigurable data center interconnect," in *Proceedings of the 2016 ACM SIGCOMM Conference*, 2016, p. 216–229.

[8] C. Avin, K. Mondal, and S. Schmid, "Demand-aware network designs of bounded degree," *Distributed Computing*, vol. 33, pp. 1–15, 2020.

[9] 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 *Proc. ACM SICOMM 2017 conference*. ACM, 2017, pp. 267–280.

[10] W. M. Mellette, R. Das, Y. Guo, R. McGuinness, A. C. Snoeren, and G. Porter, "Expanding across time to deliver bandwidth efficiency and low latency," in *Proc. 17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*, 2020, pp. 1–18.

[11] C. Griner, J. Zerwas, A. Blenk, M. Ghobadi, S. Schmid, and C. Avin, "Cerberus: The power of choices in datacenter topology design (a throughput perspective)," *Proc. ACM Meas. Anal. Comput. Syst.*, vol. 5, no. 3, dec 2021.

[12] J. ZERWAS, C. Gyorgyi, A. Blenk, S. Schmid, and C. Avin, "Duo: A high-throughput reconfigurable datacenter network using local routing and control," in *ACM SIGMETRICS*, 2023.

[13] V. Addanki, C. Avin, and S. Schmid, "Mars: Near-optimal throughput with shallow buffers in reconfigurable datacenter networks," in *ACM SIGMETRICS (accepted)*, 2023.

[14] C. Avin, K. Mondal, and S. Schmid, "Demand-aware network design with minimal congestion and route lengths," *IEEE/ACM Transactions on Networking*, vol. 30, no. 4, pp. 1838–1848, 2022.

[15] C. Avin, A. Hercules, A. Loukas, and S. Schmid, "rdan: Toward robust demand-aware network designs," *Information Processing Letters*, vol. 133, pp. 5–9, 2018.

[16] M. Pacut, W. Dai, A. Labbe, K.-T. Foerster, and S. Schmid, "Improved scalability of demand-aware datacenter topologies with minimal route lengths and congestion," *ACM SIGMETRICS Performance Evaluation Review*, vol. 49, no. 3, pp. 35–36, 2022.

[17] A. Singla, A. Singh, K. Ramachandran, L. Xu, and Y. Zhang, "Proteus: a topology malleable data center network," in *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*. ACM, 2010.

[18] 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 *Proc. ACM SICOMM Conference*, 2017, p. 295–308.

[19] C. Avin and S. Schmid, "Renets: Statically-optimal demand-aware networks," in *Proc. SIAM Symposium on Algorithmic Principles of Computer Systems (APOCS)*, 2021.

[20] S. Kandula, S. Sengupta, A. Greenberg, P. Patel, and R. Chaiken, "The nature of data center traffic: measurements & analysis," in *Proc. 9th ACM SIGCOMM conference on Internet measurement*, 2009, pp. 202–208.

[21] A. Roy, H. Zeng, J. Bagga, G. Porter, and A. C. Snoeren, "Inside the social network's (datacenter) network," in *ACM SIGCOMM Computer Communication Review*, vol. 45, no. 4. ACM, 2015, pp. 123–137.

[22] L. Feuilloley and P. Fraigniaud, "Survey of distributed decision," 2016.

[23] M. Wattenhofer and R. Wattenhofer, "Distributed weighted matching," in *Distributed Computing*, 2004.

[24] H. Ballani, P. Costa, R. Behrendt, D. Cletheroe, I. Haller, K. Jozwik, F. Karinou, S. Lange, K. Shi, B. Thomsen *et al.*, "Sirius: A flat datacenter network with nanosecond optical switching," in *Proc. ACM SIGCOMM 2020 Conference*, 2020, pp. 782–797.

[25] 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," *ACM SIGCOMM Comput. Commun. Rev. (CCR)*, vol. 41, no. 4, pp. 339–350, 2011.

[26] M. Zhang, J. Zhang, R. Wang, R. Govindan, J. C. Mogul, and A. Vahdat, "Gemini: Practical reconfigurable datacenter networks with topology and traffic engineering," 2021.

[27] Y. J. Liu, P. X. Gao, B. Wong, and S. Keshav, "Quartz: A new design element for low-latency dcns," *SIGCOMM Comput. Commun. Rev. (CCR)*, vol. 44, no. 4, pp. 283–294, Aug. 2014.

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

[29] S. B. Venkatakrishnan, M. Alizadeh, and P. Viswanath, "Costly circuits, submodular schedules and approximate carathéodory theorems," *Queueing Systems*, vol. 88, no. 3-4, pp. 311–347, 2018.

[30] G. Wang, D. G. Andersen, M. Kaminsky, K. Papagiannaki, T. Ng, M. Kozuch, and M. Ryan, "c-through: Part-time optics in data centers," *ACM SIGCOMM Comput. Commun. Rev. (CCR)*, vol. 41, no. 4, pp. 327–338, 2011.

[31] R. Schwartz, M. Singh, and S. Yazdanbod, "Online and Offline Algorithms for Circuit Switch Scheduling," in *39th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2019)*, 2019.

[32] 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 *Proc. 14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, USA, 2017, pp. 577–593.

[33] M. Hampson, "Reconfigurable optical networks will move supercomputerdata 100x faster," in *IEEE Spectrum*, 2021.

[34] 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 Transactions on Networking (TON)*, vol. 22, no. 2, pp. 498–511, 2014.

[35] F. Douglis, S. Robertson, E. Van den Berg, J. Micallef, M. Pucci, A. Aiken, M. Hattink, M. Seok, and K. Bergman, "Fleet—fast lanes for expedited execution at 10 terabits: Program overview," *IEEE Internet Computing*, vol. 25, no. 3, pp. 79–87, 2021.

[36] N. Hamedazimi, Z. 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 *ACM SIGCOMM Comput. Commun. Rev. (CCR)*, vol. 44, no. 4. ACM, 2014, pp. 319–330.

[37] 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 Comput. Commun. Rev. (CCR)*, vol. 42, no. 4, pp. 443–454, 2012.

[38] S. Kandula, J. Padhye, and P. Bahl, "Flyways to de-congest data center networks," in *Proc. ACM Workshop on Hot Topics in Networks (HotNets)*, 2009.

[39] D. Amir, N. Saran, T. Wilson, R. Kleinberg, V. Shrivastav, and H. Weatherspoon, "Shale: A practical, scalable oblivious reconfigurable network," ser. ACM SIGCOMM '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 449–464.

[40] M. Y. Teh, Z. Wu, and K. Bergman, "Flexspander: augmenting expander networks in high-performance systems with optical bandwidth steering," *IEEE/OSA Journal of Optical Communications and Networking*, vol. 12, no. 4, pp. B44–B54, 2020.

[41] T. Benson, A. Akella, and D. A. Maltz, "Network traffic characteristics of data centers in the wild," in *Proc. ACM SIGCOMM conference on Internet measurement*. ACM, 2010, pp. 267–280.

[42] Q. Zhang, V. Liu, H. Zeng, and A. Krishnamurthy, "High-resolution measurement of data center microbursts," in *Proc. 2017 Internet Measurement Conference*, ser. IMC '17, 2017, pp. 78–85.

[43] S. Zou, X. Wen, K. Chen, S. Huang, Y. Chen, Y. Liu, Y. Xia, and C. Hu, "Virtualknotter: Online virtual machine shuffling for congestion resolving in virtualized datacenter," *Computer Networks*, vol. 67, pp. 141–153, 2014.

[44] S.-E. Huang and H.-H. Su, "(1-ϵ)-approximate maximum weighted matching in poly(1/ϵ, log n) time in the distributed and parallel settings," in *Proceedings of the 2023 ACM Symposium on Principles of Distributed Computing*, ser. PODC '23, 2023.

[45] D. G. Harris, "Distributed local approximation algorithms for maximum matching in graphs and hypergraphs," in *2019 IEEE 60th Annual Symposium on Foundations of Computer Science (FOCS)*, 2019.

[46] M. Ahmadi, F. Kuhn, and R. Oshman, "Distributed Approximate Maximum Matching in the CONGEST Model," in *32nd International Symposium on Distributed Computing (DISC 2018)*, 2018.

[47] A. Panconesi and M. Sozio, "Fast primal-dual distributed algorithms for scheduling and matching problems," *Distributed Comput.*, vol. 22, no. 4, 2010.

[48] M. Fischer, "Improved deterministic distributed matching via rounding," in *31st International Symposium on Distributed Computing, DISC 2017, October 16-20, 2017, Vienna, Austria*, 2017.

[49] O. Peres and C. Avin, "Distributed demand-aware network design using bounded square root of graphs," in *INFOCOM 2023 - IEEE Conference on Computer Communications*, ser. Proceedings - IEEE INFOCOM, 2023.

[50] C. Lenzen, "Optimal deterministic routing and sorting on the congested clique," in *Proceedings of the 2013 ACM Symposium on Principles of Distributed Computing*, ser. PODC '13. New York, NY, USA: Association for Computing Machinery, 2013, p. 42–50.

[51] D. Gale and L. S. Shapley, "College admissions and the stability of marriage," *The American Mathematical Monthly*, vol. 69, pp. 9–15, 1962.

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

[53] V. Liu, D. Halperin, A. Krishnamurthy, and T. Anderson, "F10: A fault-tolerant engineered network," in *Proceedings of the 10th USENIX Conference on Networked Systems Design and Implementation*, ser. nsdi'13. USA: USENIX Association, 2013, p. 399–412.