ReNets: Statically-Optimal Demand-Aware Networks*

Chen Avin† Stefan Schmid‡

Abstract

This paper studies the design of *self-adjusting* datacenter networks whose physical topology dynamically adapts to the workload, in an *online* and *demand-aware* manner. We propose *ReNet*, a self-adjusting network which does not require any predictions about future demands and amortizes reconfigurations: it performs as good as a hypothetical static algorithm with perfect knowledge of the future demand. In particular, we show that for arbitrary *sparse* communication demands, *ReNets* achieve *static optimality*, a fundamental property of learning algorithms, and that route lengths in *ReNets* are proportional to existing lower bounds, which are known to relate to an *entropy* metric of the demand. *ReNets* provide additional desirable properties such as *compact* and *local* routing and flat addressing therefore ensuring scalability and further reducing the overhead of reconfiguration. To achieve these properties, *ReNets* combine multiple self-adjusting tree topologies which are optimized toward individual sources, called *ego-trees* in this paper.

1 Introduction

Modern datacenter networks rely on efficient network topologies to provide a high connectivity at low cost. Most existing datacenter networks also have in common that their topology is *fixed* and *oblivious* to the actual demand (i.e., workload or communication pattern) they currently serve. Rather, they are designed for all-to-all, or uniform, communication patterns, by ensuring properties such as (almost) full bisection bandwidth or $O(\log n)$ route lengths between *any* node pair in a constant-degree n-node network. However, empirical studies show that traffic patterns in datacenters are far from uniform, but rather *skewed and bursty* [1, 2, 3, 4, 5], featuring much (spatial and temporal) locality. This makes demand-oblivious networks inefficient compared to optimal designs.

This paper investigates *demand-aware* networks (DANs) designs: networks which optimize their physical topology toward the demand they serve. In particular, we consider the design of DANs which provide short average route lengths by accounting for locality in the demand and by locating frequently communicating node pairs (e.g., a pair of top-of-the-rack switches) topologically closer. Shorter routes can improve network performance (e.g., latency) and reduce costs (e.g., load, energy consumption) [6].

DANs come in two flavors: *fixed* and *self-adjusting*. Fixed DANs can exploit *spatial* locality in the demand. It has recently been shown that a fixed DAN can provide average route lengths in the order of the (conditional) *entropy* of the demand [7, 8, 9], which can be, *for specific demands*, much lower than the $O(\log n)$ route lengths provided by demand-*oblivious* networks. However, fixed DANs require *a priori* knowledge of the demand.

On the contrary, *self-adjusting* DANs do not require such knowledge and can additionally exploit *temporal* locality, by adapting the topology to the demand in an *online* manner. The vision of such self-adjusting networks is enabled by emerging optical technologies which allow us to quickly *reconfigure* the topology over time [1, 10, 11, 12, 13, 14, 15, 16, 17].

For the upper networking layers, the advantages of demand-aware vs. demand-oblivious algorithms are already well-known and widely-used, for example CDNs use demand-aware algorithms on the application layer, TCP is demand-aware on the transport layer, and traffic engineering provides demand-awareness on the network layer. However, the design of self-adjusting physical-layer topologies is still not well-understood and challenging: while more frequent reconfigurations allow to adapt the topology to the demand in a more fine-grained manner, such reconfigurations also come at a cost (e.g., related to recomputations, delays, packet reorderings [6]). Hence, an optimal tradeoff between the benefits and the costs of such reconfigurations has to be found. This tradeoff is not accounted for today: most existing solutions in the literature, e.g., [1, 18, 19], recompute demand-aware networks *periodically* and/or *from scratch*, for a (predicted) new demand matrix. Depending on the frequency of such reconfigurations, such periodic adjustments introduce unnecessary overheads or result in a poor reactivity in case of bursty demands.

Further challenges are introduced by the online nature of the problem and the lack of a priori knowledge

*Supported by European Research Council (ERC) Consolidator project, Self-Adjusting Networks (AdjustNet), grant agreement No. 864228, Horizon 2020, 2020-2025.

†School of Electrical and Computer Engineering, Ben Gurion University of the Negev, Israel.

‡Faculty of Computer Science, University of Vienna, Austria.

about the demand. Ideally, a self-adjusting network provides an optimal performance *even in hindsight*: despite the lack of information on the demand, the performance is at least as good as the performance of a hypothetical (fixed) demand-aware network with a priori knowledge of the demand, for sufficiently long demands. This property is called *static optimality*: static optimality is a strong notion of optimality and frequently used to evaluate algorithms based on limited information, for example in the context of coding (e.g., dynamic Huffman codes [20]), self-adjusting datastructures (e.g., splay trees [21]), or repeated games and machine learning [22, 23, 24, 25].

In order to ensure scalability, it is also important that self-adjusting networks support *compact routing*, i.e., small routing tables, and also limit the impact of topological reconfigurations on the routing tables and the routing algorithm, i.e., do not require the recomputation of routes.

1.1 Our Contributions Our main contribution is a self-adjusting demand-aware network called *ReNet* which comes with several, theoretically proven, attractive features: (1) *ReNets* provide route lengths which are proportional to an *entropy* metric of the demand, while keeping reconfiguration costs minimal. In particular, we prove that *ReNets* are *statically optimal* under sparse communication patterns. *ReNets* are based on *online* algorithms and do not require any knowledge of future demands. (2) *ReNets* are highly scalable in that they only require a constant number of reconfigurable links per switch or router, and in that they feature *compact routing*, i.e., constant-size forwarding tables. (3) By supporting *local routing*, *ReNets* further reduce the reconfiguration costs: the effect of topological changes on forwarding tables is minimal. (4) Last but not least, *ReNets* do not require any specific addressing scheme, addresses can be arbitrary, so routing can be based on, e.g., flat, topology-independent MAC addresses or hierarchical, location-dependent IP addresses.

The design of *ReNets* relies on concepts from self-adjusting datastructures. In particular a *ReNet* is based on a set of trees, called *ego-trees*. Each ego-tree is (dynamically) optimized for an *individual*, highly active, node which acts as the root of the tree. The ego-tree of a given node stores the *working set* of that node (e.g., top-of-rack switches), its most recent communication partners. A *ReNet* is then a union of all the ego-trees of individual nodes, using algorithmic manoeuvres to make sure that the degree (and routing tables) remain constant at any time.

More specifically, the working set of each of these nodes is organized as a self-adjusting binary search tree (*BST*). While different types of such ego-trees can be used (e.g., Huffman trees, tango trees [26], etc.), *ReNet* are based on *splay trees* [21]. As we will see, this will result in desirable properties, such as compact and local routing as well as static optimality.

1.2 Scope and Limitations The main contribution of this paper is a conceptual one, in that we show which metrics and properties can be achieved theoretically by self-adjusting networks, focusing on the algorithmic foundations. In particular, we consider an online scenario where communication requests arrive over time (as it is usual in competitive analysis [27]), we do not optimize constants in our algorithm, and we ignore the impact of our algorithms on other networking layers such as the transport layer. While these aspects are important, they are left to a future dedicated study which will require additional methodologies and a specific application scenario.

1.3 Organization The remainder of this paper is organized as follows. We introduce our model for self-adjusting demand-aware networks in Section 2, and present some empirical motivation in Section 3. In Section 4 we describe the algorithms underlying ReNets, and formally analyze them in Sections 5 and 6. After reviewing related work in Section 7, we conclude and discuss future work in Section 8.

2 Preliminaries and Model

We consider a set V of n nodes $V = \{1, \ldots, n\}$ with unique but otherwise *arbitrary* addresses. The communication *demand* among these nodes is described as a (finite or infinite) sequence $\sigma = (\sigma_0, \sigma_1 \ldots)$ of *communication requests* where $\sigma_t = (u, v) \in V \times V$ is the source-destination pair that communicate at time t[1]. The communication demand is *revealed in an online manner* and can be adversarial.

In order to serve this demand, the nodes V must be inter-connected by a demand-aware network (DAN), N, defined over the same set of nodes. In the case of a *self-adjusting* DAN, N can also change over time, and we denote by N_t the network at time t. For scalability reasons and since reconfigurable links may be costly and consume space, the DAN must be chosen from the family of *degree-bounded* topologies: the networks N_t are required to be of *constant* degree at most Δ which is predefined.

The cost to serve a communication request $\sigma_t = (u, v)$ on the DAN at time t, is considered to be the hop

[1]The result and model can be extended to the case where several different sources can communicates at the same time.

distance $\mathrm{d}_{N_t}(u,v)$ from u to v over N_t, along the routing path chosen by the algorithm. If not specified otherwise, we assume shortest path routing. We are interested in the fundamental tradeoff between the benefits of self-adjusting algorithms (i.e., shorter routes) and their costs (namely reconfiguration costs). Let $\mathcal{A}$ be an algorithm that given the request σ_t and the network N_t at time t, reconfigures the current network to N_{t+1} at time $t+1$. The cost of the network reconfiguration at time t is given by the number of link changes performed to change N_t to N_{t+1}; when $\mathcal{A}$ is clear from the context, we will simply write this cost as $\mathrm{adj}(N_t, N_{t+1})$.

We can now define the cost of an algorithm $\mathcal{A}$ for self-adjusting network to serve a demand σ starting from an initial network N_0:

Average and Amortized Cost. Given an algorithm $\mathcal{A}$, an initial network N_0, a distance function $\mathrm{d}_N(\cdot)$, and a demand $\sigma = (\sigma_0, \sigma_1, \ldots, \sigma_{m-1})$ of communication requests, we define the *(average) cost* incurred by $\mathcal{A}$ as:

$$\mathrm{Cost}(\mathcal{A}, N_0, \sigma) = \frac{1}{m}\sum_{t=0}^{m-1}(\mathrm{d}_{N_t}(\sigma_t) + \mathrm{adj}(N_t, N_{t+1}))$$

The *amortized cost* of $\mathcal{A}$ is defined as the worst possible cost of $\mathcal{A}$ over all initial networks N_0 and all sequences σ, i.e., $\max_{N_0,\sigma} \mathrm{Cost}(\mathcal{A}, N_0, \sigma)$. □

In addition to the DAN, nodes can also communicate over a demand-*oblivious* network: reconfigurable datacenter networks are usually *hybrid* [1, 6], connecting fixed (electric) switches with reconfigurable (optical) switches. The demand-oblivious network plays a minor role in this paper, and is only used to exchange *control* information (e.g., discover new neighbors). Route lengths on the demand-oblivious networks cost D per request, where D is a parameter: e.g., D is the diameter of the demand-oblivious network, $D = \Theta(\log n)$ in constant-degree networks.

Let $\sigma' = \sigma[t, t+\ell]$ be a consecutive subsequence of length ℓ starting at time t. For a sequence σ, or a subsequence σ', let the *demand graph* $G(\sigma') = (V(\sigma'), E(\sigma'))$ be a directed and weighted graph as follows. The node set V of G is given by the set of nodes participating in σ', i.e., $V(\sigma') = \{v : v \in \sigma'\}$, and the set of directed edges E is given by $E(\sigma') = \{\sigma'_t : \sigma'_t \in \sigma']\}$. The weight $w(e)$ of each directed edge $e = (x, y) \in E$ is the frequency $f(x, y)$ of the request from source x to destination y in σ', where $\sum_{x,y \in V(\sigma')} f(x, y) = 1$.

Next, we define the ***empirical entropy*** of the demand $\sigma = (\sigma_1, \sigma_2, \ldots, \sigma_m)$. We will interpret σ as a *joint empirical frequency distribution* $f(\hat{X}, \hat{Y})$, where $\hat{X}$ is a random variable (r.v.) describing the empirical frequency of the sources and $\hat{Y}$ is a r.w describing the empirical frequency of the destinations. More formally, let $f(x, \cdot) = \sum_y f(x, y)$ and $f(\cdot, y) = \sum_x f(x, y)$ and $\hat{X}_\sigma = \{f(x_1, \cdot), \ldots, f(x_n, \cdot)\}$ be the empirical frequency distribution of the *communication sources*. We omit σ in $\hat{X}_\sigma$ when it is clear from the context. The *empirical entropy* $H(\hat{X})$ is then defined as $H(\hat{X}) = -\sum_{i=1}^{n} f(x_i) \log_2 f(x_i)$, where $f(x_i)$ is used as a shorthand for $f(x_i, \cdot)$. Similarly, we define the empirical entropy of the *communication destinations*, $H(\hat{Y})$, where $\hat{Y}_\sigma = \{f(\cdot, y_1), \ldots, f(\cdot, y_n)\}$. We use the normalization $f(x|y) = f(x, y)/f(\cdot, y)$. The empirical *joint* entropy $H(\hat{X}, \hat{Y})$ is defined as $H(\hat{X}, \hat{Y}) = -\sum_{i,j} f(x_i, y_j) \log_2 f(x_i, y_j)$ and the empirical *conditional* entropy $H(\hat{X}|\hat{Y})$ which measures spatial locality as $H(\hat{X}|\hat{Y}) = -\sum_j f(y_j) \sum_i f(x_i \mid y_j) \log_2 f(x_i \mid y_j)$. Note that $H(\hat{X}|\hat{Y}) = \sum_j f(y_j) H(\hat{X}_{y_j})$ where $\hat{X}_{y_j}$ is the frequency distribution of the sources for the destination y_j. We may simply write H for the entropy if the usage is given by the context. By default, the entropy H is computed using the binary logarithm; if a different logarithmic basis Δ is used to compute the entropy, we will explicitly write H_Δ.

3 Intuition & Empirical Motivation

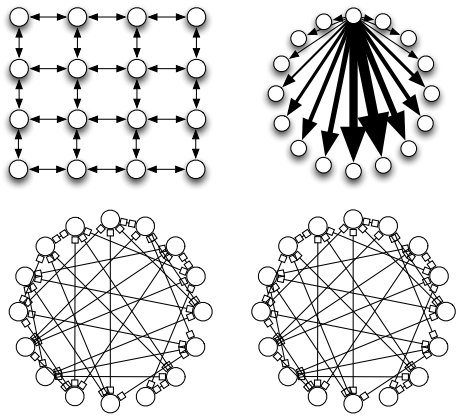
It was recently shown that the *conditional entropy* of the demand, and in particular $\max(H_\Delta(\hat{Y}|\hat{X}), H_\Delta(\hat{X}|\hat{Y}))$ is a lower bound for the average route length in any (constant) degree-Δ bounded, *fixed* network [7][2]. This bound can be (asymptotically) matched if the demand σ is sparse and is known *a priory*, before designing the network. In contrast, in this work, we are interested in solutions that match the conditional entropy lower bound, but for a demand σ that is *unknown* a priory. This makes the task much more challenging.

This section provides intuition on the connection between the entropy of the demand and route lengths in *fixed* demand-aware networks, as well as on the benefits of demand-aware networks in general (see also [9]), by giving simple examples and initial empirical results on real traces from Facebook.

Recall that the conditional entropy of a demand σ is a lower bound for the average path length even in demand-aware networks [7,8]. First, we demonstrate that the route lengths in *demand-oblivious* networks (such as state-of-the-art expander networks [28, 29]) cannot be proportional to the lower bound of the conditional entropy of the demand. Therefore, they cannot provide the desired solution (even in the presence of traffic engineering flexibilities [30]).

To give a simple example (for the sake of simplicity

[2]We note that the result in [7] is stated for the entropy and not the empirical entropy, however, the claim follows directly.



(a) (b)

Figure 1: **Expander networks do not achieve optimal average route lengths for sparse demand graphs.** (a) Oblivious embedding of a 2-dimensional grid demand graph (upper graph) on a constant degree expander network (lower graph) will result in average route lengths of $\Omega(\log n)$, while the conditional entropy of the demand graph is less than two. (b) Oblivious embedding of a weighted star-shaped demand graph on a constant degree expander network will result in an average route length of $\Omega(\log n)$ while the conditional entropy of the demand graph could be much lower

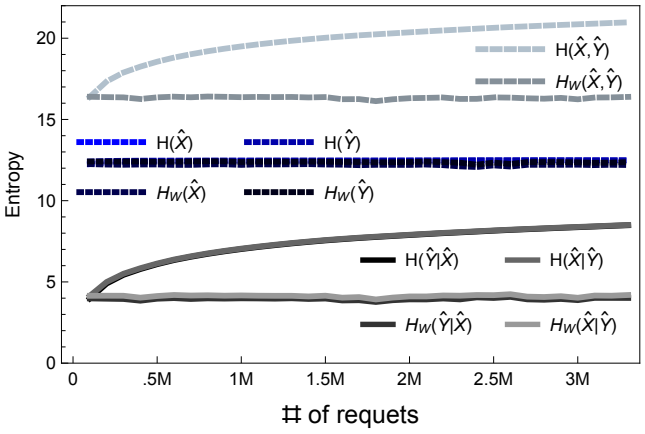


Figure 2: Entropy measures in Facebook's workload.

and clarity we leave some of the details out), consider a workload describing a communication pattern σ whose demand graph $G(\sigma)$ forms a two-dimensional square grid, of size $\sqrt{n} \times \sqrt{n}$, see Figure 1 *(a)*. For this sequence σ, $H(\hat{X})$ and $H(\hat{Y})$ are of order $\log n$, since the frequency of sources and destinations is uniform which results in *the maximum possible entropy*. If we serve this σ on a static expander in an demand-oblivious way (i.e., node locations are random, or arbitrary in the network) it will result in an average route length in the order of $\log n$, which is the diameter of a bounded degree expander. However, since every node has at most four neighbors, the conditional entropy of σ (both $H(\hat{Y}|\hat{X})$ and $H(\hat{X}|\hat{Y})$) is only *2*: a gap of $\Theta(\log n)$. A demand-aware network design (of the same bounded degree) could potentially achieve the lower bound of the conditional entropy.

Another simplified example introducing a large gap of $\Theta(\log n)$ between demand-oblivious and demand-aware networks is a demand graph $G(\sigma)$ which forms a star (with *unbounded* degree), and where node pairs communicate at different frequencies described by a highly skewed distribution, see Figure 1 *(b)*. For such a skewed demand, the conditional entropy could be much lower than $\log n$ which will be the cost of serving this demand on an demand-oblivious expander.

More generally, one can see that every sparse communication pattern which is embedded on a demand-oblivious expander (e.g., via a random embedding on an expander), will result in average route lengths in the order of $\Omega(\log n)$, the diameter, regardless of the entropy or the conditional entropy of the demand.

So, the potential benefit of demand-aware networks boil down to the *structure* of the demand, and in particular its empirical entropies. In order to illustrate the optimization potential, we can use the empirical traces made available by Facebook for their datacenter [3] as an example. In Figure 2 we plot the empirical and the conditional entropy, $H(\hat{X})$ and $H(\hat{X}|\hat{Y})$, respectively. The demand σ here consists of $n = 13748$ communication partners. Note that $\log n = 13.74$ for this case (we consider the binary logarithm). The figure considers times t that use multiplicatives of $100K$ requests. For each time t, the measures are presented both for the full range $\sigma[1, t]$ (labelled H) as well as for a time window of the last $100K$ requests $\sigma[t - 100K, t]$ (labelled H_W), to shed light on the *temporal locality*. Clearly the conditional entropy is lower than the entropy, and in particular the conditional entropy of the window is much less than the entropy of X and Y. This indicates that demand-aware designs could reduce the average route length in the network even for real-life communication traces.

4 ReNet: A Statically-Optimal Online Demand-Aware Network

This section presents *ReNet*s a statically optimal algorithms for self-adjusting networks of bounded degree with a number of desirable properties. We first introduce these properties, then present algorithmic building blocks, and finally describe *ReNet*s' algorithms and for-

warding tables in details.

4.1 Desirable Properties The main challenge faced by self-adjusting DANs is that information about (future) demand may not be available. Our goal is to design algorithms for *statically optimal* networks:

Static Optimality. Let STAT be an optimal algorithm to design a static network with a perfect knowledge of the demand σ, and let ON be an online algorithm producing a dynamic degree-bounded network (i.e., the maximum degree is at most Δ). We say that ON is *statically optimal* if, for sufficiently long communication patterns σ:

$$\rho = \max_{\sigma} \frac{\text{Cost}(\text{ON}, \emptyset, \sigma)}{\text{Cost}(\text{STAT}, N^*, \sigma)}$$

is *constant*. Here, $N_0 = \emptyset$ is the empty network from which ON starts, and N^* is the statically optimal degree-bounded network for σ. In other words, ON's cost is at most a constant factor higher than STAT's in the worst case. □

Static optimality is a very strong property that guarantees an (asymptotically) optimal performance *even in hindsight*.

In order to be scalable and in order to minimize reconfiguration costs, self-adjusting networks must provide some additional fundamental properties. In particular, we require that each node in a self-adjusting network not only needs at most a constant number of reconfigurable links, but we would like to have an even stronger property, namely *compact routing*: that the forwarding tables are small, i.e., of constant size [31, 32].

Compact Routing. A network supports *compact routing* if the sizes of the nodes' forwarding tables are constant, i.e., independent of the network size. □

A key challenge in the design of self-adjusting networks is that topological reconfigurations may negatively affect routing. A particularly attractive (but seemingly difficult to achieve) property that enables efficient routing in dynamic networks is *local routing*:

Local Routing. A network provides *local routing* if packets can be forwarded based on local knowledge only. □

In particular, local routing provides an efficient alternative to many existing routing schemes in that topological changes can be reflected efficiently in the forwarding tables, and do not entail the global recomputation of routes.

Efficient reconfigurations are further enabled by a topology and location independent addressing scheme:

Arbitrary Addressing. Nodes can have arbitrary (but unique) addresses. □

In fact, the results derived in this paper apply to *any* addressing scheme, e.g., based on location-independent MAC addresses or location-dependent IP addresses.

The main result of the paper regards the static optimality of *ReNets* for sparse demand that we define next.

(c, δ)-sparse Communication. We call a communication demand σ (c, δ)-*sparse* if and only if any subsequence σ' of σ of length $|\sigma'| \leq \delta$, involves no more than $c \cdot n$ unique communication pairs where c is a constant and δ is a function of n. That is, σ' implies a *sparse demand graph* $G(\sigma') = (V, E(\sigma'))$ of average degree $2|E(\sigma')|/n \leq 2c$. □

Note that for $\delta = \infty$, the entire communication pattern σ needs to be sparse. For $\delta \leq cn$, the constraint is trivial.

4.2 Algorithmic Building Blocks We describe *ReNets* using a top-down approach from concepts to details. We start with the main ideas and building blocks of *ReNets*. Ideally, each node $u \in V$ in *ReNet* connects *directly* to all its communication partners in $G(\sigma)$, achieving an ideal average route length of 1. However, this is infeasible, as (1) the communication partners are not known to u *a priori* and (2) a node u may have *many* communication partners (even in an otherwise sparse demand graph), which would result in a high degree and large forwarding tables. To overcome this, *ReNet* leverages several key concepts:

Concept 1 - Working Set. Each node u in *ReNet* keeps track of it recent active communication partners, i.e., the so-called *working set* $\mathcal{W}(u)$, hoping to exploit temporal locality as they are also likely to be relevant in the near future. The working set will be defined over the recent subinterval of σ that will be defined later.

Concept 2 - Small and Large Nodes. A node u in *ReNet* pursues one of two different approaches to communicate with its recent communication partners, depending on its working set size. Towards this end, we define the *size* of a node u to be the cardinality $|\mathcal{W}(u)|$ of u's working set. We say that a node u is *small* if the size of u is smaller than a parameter θ and otherwise, a node is called *large*. For now assume θ is a constant which depends on the sparsity of the communication sequence, we will discuss the details later. A small node will communicate to its communication partners *directly*; a large node *indirectly*, by forwarding the traffic along its *ego-tree*, which we explain next.

Concept 3 - Ego-Tree/Ego-BST. For large nodes u, establishing links or storing forwarding rules for each communication partner in $\mathcal{W}(u)$ is infeasible as it would result in forwarding tables of non-constant size. Thus, in *ReNet*, a *large* node u organizes its communication partners $\mathcal{W}(u)$ in a self-adjusting *ego-tree*: a *(tree)*

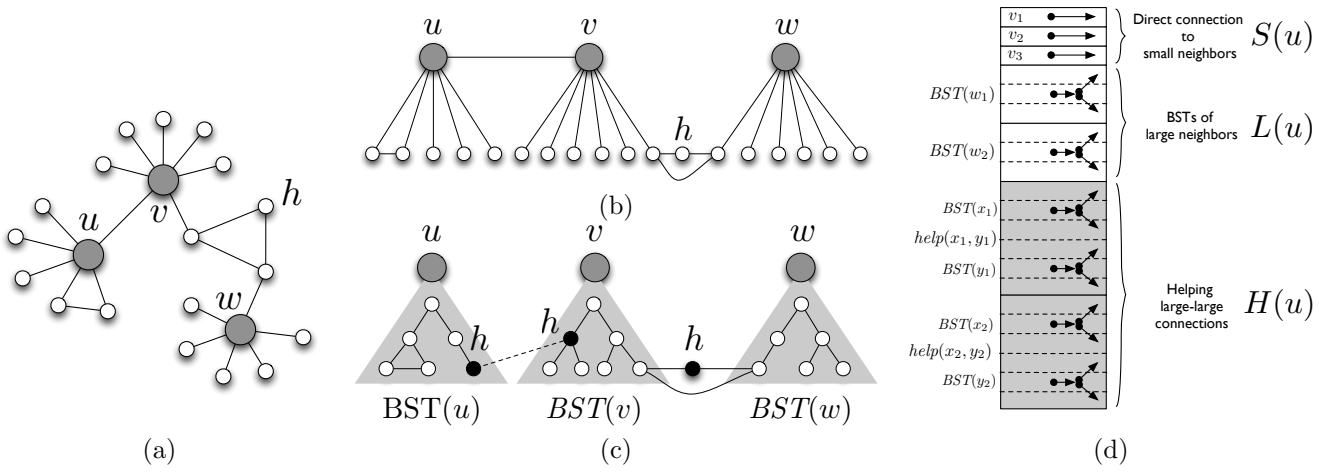


Figure 3: Design principles of a (fixed and adaptive) *ReNet*: (a) The sparse demand graph $G(\sigma)$. Nodes are divided into large (*gray*, e.g., u, v, w) and small (*white*, e.g. h) nodes. (b) Hierarchical representation of the demand graph. Problematic edges are edges between two large nodes (e.g., (u, v)). (c) *ReNet*: every large node x has an *ego-BST*(x) connecting its *working set* $\mathcal{W}(x)$. Every large-large edge, is routed with the help of a small node (acting as relay between *ego-BST*s, *black*). E.g., h is the helping node for edge (u, v) and participates in *ego-BST*(u) as a relay node toward v and in *ego-BST*(v) as a relay toward u. The resulting network has bounded degree. (d) Forwarding table for a *small* node u, see text.

network optimized for just this source. In particular, we propose to use a self-adjusting *binary search tree* for the ego-tree of a large node u, short *ego-BST*(u). An important property of BSTs (both fixed and self-adjusting) is that they naturally support local and compact routing for messages to/from the (root) of the tree. In particular, the (self-adjusting) *ego-BST*(u), is used to efficiently *store* and *lookup* (i.e., forward to) neighbors $v \in \mathcal{W}(u)$. Each node that belongs to such an *ego-BST*(u) supports the following interface:

- *ego-BST*(u).insert(v): insert v to *ego-BST*(u)
- *ego-BST*(u).forward(v): forward packet toward v
- *ego-BST*(u).adjust(): local update of tree network datastructure

ReNet uses *splay trees* [21] as their *self-adjusting ego-BST*s. An additional nice property of splay trees is that they are known to be statically optimal.

Concept 4 - Self-Adjustments. *ReNets* perform two types of self-adjustments. In order to update the neighborhood structures and optimize the network after a routing request, a node u makes use of the adjust() operation of its tree. For example in splay trees, we issue a *splay* operation on the tree network. Second, *ReNets* keep track of the total size of the nodes' working sets. Once the total size, $\sum_v \mathcal{W}(v)$, exceeds $n \cdot \theta/2$, all working sets are *cleared* (in the spirit of *flush-when-full* or *marking* techniques known from competitive paging [33]). Such reset operations are necessary to follow temporal locality, allowing the nodes to update the working sets and hence be able to adjust to changing demand patterns.

Concept 5 - Helper Nodes. The problem with the approach described so far is that while nodes in a single BST are of degree at most three (*parent*, *left child*, *right child*), a *large* node v can still appear in multiple trees beside its own tree if it has *large* nodes in its working set. Combined, these trees can induce a large forwarding table on v, and hence, an additional mechanism is needed to bound the degree. To this end, *ReNets* leverage small nodes to help two communicating large nodes keep the forwarding table small. Concretely, as long as the average node size is smaller than $\theta/2$, a *ReNet* exploits small nodes (of size below θ) as *helper nodes*. For example, if a small node h serves as *relay* between two communicating large nodes v and u, node h will appear in *both* trees networks *ego-BST*(v) and *ego-BST*(u): in *ego-BST*(v), h will serve as a forwarder toward u and in *ego-BST*(u), as a forwarder toward v. See Figure 3 for an example. As we will discuss later, this not only allows us to bound the size of the forwarding table, but also to preserve local routing.

Concept 6 - Centralized Bookkeeping and Coordination. While reconfiguration is decentralized, bookkeeping and coordination is centralized in *ReNet*. This avoids complexities due to possible inconsistencies and is efficient: a network coordinator (e.g., an arbitrary node in the network) only needs to keep track of which nodes are *large* and which nodes are *small*. That is, nodes

inform the coordinator when they need to add a new partner to their working sets. Given this information, the coordinator can assign helper nodes upon request, in an event-driven manner. If no helper nodes are left, the coordinator schedules a reset() operation that clears all working sets for all nodes, and sets their size to *small*. Such a reset can be done using a spanning tree, at linear cost.

Figure 3 illustrates some of the concepts introduced above, such as small and large nodes, working set, *BST*, and helping nodes.

4.3 Details of Forwarding Table and Reconfiguration Algorithm With these intuitions in mind, we now present the network reconfiguration and forwarding algorithms underlying *ReNets* in details.

4.3.1 Forwarding Table and Reconfigurable Ports Each node $u \in V$ maintains the following forwarding table of given *constant* size 6θ (details on this size will become clear later) whose content may change over time. Each entry in the table leads to a reconfigurable link (port) that can connect to any other node with available port. W.l.o.g. links are bidirectional and messages can be sent both ways.

If u is *small*, u's forwarding table contains (Figure 3 (d)):

- A set $S(u) = \{v_1, v_2, \ldots\}$ of *small* neighbors of u. For each v_i, u has a *direct, physical* link (port) toward v_i.
- A set $L(u) = \{\textit{ego-BST}(w_1), \textit{ego-BST}(w_2), \ldots\}$ of *ego-BSTs* of *large* neighbors of u. In each of these trees, say *ego-BST*(w), u will participate and forward messages toward/from the root of the tree, w, including messages where u is the source/destination. Each such tree requires three entries in the forwarding table, and three *physical ports* (i.e., direct links): (1) a forward entry to the parent of u in *ego-BST*(w), (2) a forward entry to the left child of u in *ego-BST*(w), and (3) similarly for the right child in *ego-BST*(w).
- A set $H(u) = \{(x_1, y_1), (x_2, y_2), \ldots\}$ of pairs of *large* nodes x_i, y_i for which u acts as a *helper*. Helping such a *large-large* connection, requires six entries in the forwarding table and six ports: three entries and ports for each tree, *ego-BST*(x_i) and *ego-BST*(y_i).

If u is *large*, u's forwarding table is simpler and contains:

- A (physical) link to the current root of *ego-BST*(u).
- A set of at most $6\theta - 1$ virtual roots to improve the performance of *ego-BST*(u).[3]

Algorithm 1 Source u, upon request $u \to v$

```
1: if u is large then
2:    forward to root of ego-BST(u)
3: else
4:    (* small node *)
5:    if v ∈ S(u) then
6:       forward directly to v
7:    else if ego-BST(v) ∈ L(u) then
8:       ego-BST(v).forward(v) (to parent of u)
9:    else
10:      (* new partner *)
11:      notify coordinator: addRoute(u → v)
```

Algorithm 2 Destination v, upon request $u \to v$

```
1: process packet
2: if request received on some ego-BST(w) then
3:    ego-BST(w).adjust()
```

Note that for each node u, its forwarding table contains its working set, $\mathcal{W}(u)$, among some additional information.

4.3.2 Roles The algorithms underlying *ReNets* involve four different node roles:

- **The Source** (Algorithm 1): Let u be the source of a communication request (u, v). In case u is a large node, it will simply forward the request to the root of *ego-BST*(u) (or directly to v if it is one of the virtual roots of *ego-BST*(u)). In case u is a small node then: if $v \in S(u)$ then it will forward it directly to v; else, if u participates in *ego-BST*(v) then u will forward it to its parent in *ego-BST*(v). Else, if v is not in the working set of u, $\mathcal{W}(u)$, it is a new communication partner, then u will notify the coordinator and request being connected to v via the procedure *addRoute*$(u \to v)$, and to add v to $\mathcal{W}(u)$.
- **The Destination** (Algorithm 2): The behavior of the destination v of a given communication request (u, v) is simple: it delivers the request to the upper layer and if needed, triggers an adjust() operation on the *ego-BST*(w), for which the packet was delivered:

[3] The use of virtual roots is a practical optimization. In a traditional self-adjusting BST, the root changes over time, depending on the demand: accessed elements are moved to the root. In *ReNet*, a node u uses a set of virtual pointers to implement the root of *ego-BST*(u): the root is implemented using a *constant* set of nodes (all at distance 1), managed in a queue, evicting the least-recently used (lru) root. However, this optimization does not affect the asymptotic performance of our network.

Algorithm 3 Forwarder x, for request $u \to v$

Require: by definition $x \in$ *ego-BST*(u) and/or $x \in$ *ego-BST*(v)

```
1: if x ∈ ego-BST(v) then
2:     ego-BST(v).forward(v) (to parent of x)
3:   if x ∈ ego-BST(u) then
4:     (* x is an helper to (u, v) *)
5:     ego-BST(u).adjust()
6: else
7:   (* x ∈ ego-BST(u) *)
8:   if ∃ child x toward v then
9:     ego-BST(u).forward(v) (to child)
10:  else
11:    notify coordinator: addRoute(u → v)
```

this optimizes the network to account for recent communications. Note that a message can reach v either directly, via *BST*(v) or via *BST*(u).

- **The Forwarder** (Algorithm 3): A forwarder x is a node which is neither the source nor the destination of a communication request (u, v), i.e., $u \neq x \neq v$. It acts as an inner node in the ego-tree network and may also be a helper (*see below*). By our construction, node x must hence either be part of *ego-BST*(u) or *ego-BST*(v), *or both* (if it is a helper). If x is part of *ego-BST*(v) of the destination, it needs to forward the request toward the root v; else if x is part of the *ego-BST*(u) of the source, it needs to forward the request to the correct child based on the ID of v. If x is a helper, it belongs to both *ego-BST*(u) and *ego-BST*(v), and additionally needs to initiate *ego-BST*(u).adjust() to update the tree. If x has no child toward v in *ego-BST*(u) then it notifies the coordinator to *addRoute*$(u \to v)$.
- **The Coordinator** (Algorithm 4): The coordinator keeps track of which nodes are small, which nodes are large, and which small nodes have room in their forwarding table to help large-to-large edges. To serve an *addRoute*$(u \to v)$ request, the coordinator distinguishes between different cases, potentially resetting the forwarding tables (using *reset*(), see below), adding helper nodes where needed or rendering the source and/or destination node large (using *makeLarge*(), see below). In the simplest case, both u and v are small and the coordinator can instruct the two nodes to connect directly. If one node is large and one small, the route request is served by inserting one node in the other node's *ego-BST*. Only if both nodes are large, the coordinator finds a helper node, x, which is used to relay between the two *ego-BST*s, which must already exist. x is then added to *ego-BST*(u) as v and to *ego-BST*(v) as u.

  When the coordinator learns that a node u needs to become large, it invokes the *makeLarge*(u) method, which instructs the creation of *ego-BST*(u). On this occasion, the coordinator iterates over the working set of u: in case of a small neighbor v, v is inserted into the *ego-BST*(u) directly; otherwise, a new helper node is used.

  The coordinator also instructs the nodes to reset their working sets and forwarding tables if no more helper nodes are available, i.e., if the *total* sizes of the working sets are $\sum_u \mathcal{W}(u) \geq n\theta/2$ and the network is, what we call, *full*. Concretely, using the *reset*() method, the coordinator instructs all nodes to clear their forwarding tables (i.e., working sets).

5 Analysis of Static Optimality

We now present a formal analysis of the properties and performance of *ReNets*. To improve readability, some lemmas and proofs are deferred to Section 6. We call a communication sequences σ *sparse* if it is (c, δ)-sparse for a constant c and $\delta = \Omega(cnD)$, where D is an upper bound on a single request's routing cost on the demand-*oblivious* network (henceforth usually assumed to be $\Theta(\log n)$, the minimum possible diameter for a scalable, constant-degree network).

We now show that forwarding in *ReNets* does not require a global routing algorithm and can use arbitrary addressing, and its size is bounded by a constant $\Delta = 6\theta = 24c$.

THEOREM 5.1. *For any sequence σ,* ReNet*s provide Δ-compact and local routing, as well as arbitrary addressing.*

Proof. Local routing. The proof of the local routing property is by construction and the states of the source u and the destination v. For routing a request from a small node u to a small node v, the packet is directly forwarded to the destination v. For routing a request from a small node u to a large node, the packet is forwarded to v by traversing *ego-BST*(v) from parent to parent, until the root of *ego-BST*(v) is reached, and from there directly to v. For routing a request from a large node u to a small node v, the packet is forwarded to v by traversing *ego-BST*(u) from the root of *ego-BST*(u) downward to v, similar to a classic search on a binary search tree which is *local*: no global information is needed, only the information to which child to forward, based on the destination ID. To maintain the BST tree and its properties, we only need a total order on the ID space (i.e., address) of the nodes. For routing a request from a large node u to a large node v, the packet is

Algorithm 4 Coordinator

addRoute$(u \rightarrow v)$:

1: **if** total size of all working sets is $\sum_u \mathcal{W}(u) \geq \theta n/2$ **then**
2: (* network is *full* *)
3: reset()
4: (* else: available helper x must exist *)
5: add v to the working set of u, $\mathcal{W}(u)$
6: **if** u is small but $|\mathcal{W}(u)| = \theta + 1$ **then**
7: ***makeLarge***(u)
8: add u to the working set of v, $\mathcal{W}(v)$
9: **if** v is small but $|\mathcal{W}(v)| = \theta + 1$ **then**
10: ***makeLarge***(v)
11: (* available room in both tables to add edge $(u \rightarrow v)$ *)
12: **cases:**
13: **if** u and v small:
14: u connects directly to v, update $S(u), S(v)$
15: **if** u small and v large:
16: *ego-BST*(v).insert(u), update $L(u)$
17: **if** u is large and v is small:
18: *ego-BST*(u).insert(v), update $L(v)$
19: **if** u is large and u is large:
20: find a helper node x
21: *ego-BST*(u).insert$(x$ as $v)$
22: *ego-BST*(v).insert$(x$ as $u)$

makeLarge(u):

1: create *ego-BST*(u)
2: **for** each $v \in \mathcal{W}(u)$ **do**
3: **if** v is small **then**
4: *ego-BST*(u).insert(v)
5: **if** u is large **then**
6: find a helper node x
7: *ego-BST*(u).insert$(x$ as $v)$
8: *ego-BST*(v).insert$(x$ as $u)$

reset():

1: **for** each $u \in V$ **do**
2: **inform** to clear $S(u)$, $L(u)$, $\mathcal{W}(u)$
3: **set** u to small

forwarded to v in two steps. By construction, there must exist a helper node x that participates both in *ego-BST*(u) and *ego-BST*(v). First the request is forwarded on *ego-BST*(u) downward to x (which is stored in the tree as v). Then, x notes that the destination is v and forwards the packet upward to v, on *ego-BST*(v). Since all binary search trees in the system are maintained locally using the adjust() method, no global routing algorithm is needed.

Compact routing. We set the threshold θ to be twice the largest possible average degree in a window of size at most δ. Since σ is (c, δ)-sparse we have $\theta = 4c$, and every node with working set size less than θ is *small*, and otherwise, it is *large*. Let $\Delta = 6\theta$ (a constant) and we set the forwarding table to size Δ, so a *ReNet* supports compact routing. We need to show that the forwarding table does not exceed the size Δ. As long as the coordinator did not call reset(), for a *large* node the forwarding table is by design at most Δ: it contains one link to its *ego-BST* root and a set of links to at most $\Delta - 1$ virtual roots. For a *small* node we prove this in Lemma 6.3, that we will state later.

Arbitrary addressing. The support for arbitrary addressing follows by design, since the search operation in taken from binary search trees and can support it naturally. ∎

We can now prove our main result: that *ReNet* are statically optimal.

THEOREM 5.2. *For any (c, δ)-sparse communication sequence σ, where $|\sigma| \geq \delta = \Omega(nD)$, there is a constant Δ for which* ReNet*s are statically optimal for Δ-degree bounded networks.*

Proof. We again set the threshold to be twice the average degree $\theta = 4c$, let $\Delta = 6\theta$ (a constant), and set the forwarding table to size Δ. Let N^* be the optimal Δ-degree bounded network used by the optimal static algorithm STAT(σ). From [7] it follows that the average cost of STAT is lower bounded by the conditional entropy:

$$\text{Cost}(\textsc{Stat}, N^*, \sigma) \geq \Omega\left(\max(H_\Delta(\hat{Y}_\sigma \mid \hat{X}_\sigma), H_\Delta(\hat{X}_\sigma \mid \hat{Y}_\sigma))\right)$$

We will prove static optimality of *ReNets* in two steps. First, we will show that the routing cost of a *ReNet* is optimal and proportional to its trees adjusting cost. Second, we will bound the cost of the operations and messages that are related to the coordinator in the *ReNet*. Overall, we will show that the amortized cost of a *ReNet* is

$$\text{Cost}(\textit{ReNet}, \emptyset, \sigma) \leq O\left(H(\hat{Y}_\sigma \mid \hat{X}_\sigma) + H(\hat{X}_\sigma \mid \hat{Y}_\sigma)\right)$$

making it order optimal since Δ is constant (recall that $N_0 = \emptyset$ is an empty initial network).

We divide σ into subsequences, $\sigma^{(i)}$, separated by the i'th call to the reset() operations announced by the coordinator. If no reset() is called then $\sigma^{(1)} = \sigma$. If reset() was called k times then the last (partial) subsequence is denoted by $\sigma^{(k+1)}$. We start with the analysis of a single "window", $\sigma^{(1)}$, which is the subsequence of σ from the start until the first reset()

operation. The length of $\sigma^{(1)}$ is $\Omega(cnD)$ by assumption on sparsity and it contains exactly cn unique requests (see the Coordinator algorithm, Algorithm 4). We claim the following:

LEMMA 5.1. ReNet *is statically optimal on* $\sigma^{(1)}$.

Proof. Let

$$H^{(1)}_{\text{con}} = \max(H(\hat{Y}_{\sigma^{(1)}} \mid \hat{X}_{\sigma^{(1)}}), H(\hat{X}_{\sigma^{(1)}} \mid \hat{Y}_{\sigma^{(1)}}))$$

be the maximum of the conditional entropies. We need to show that *ReNet*'s cost on $\sigma^{(1)}$ is $O(H^{(1)}_{\text{con}})$ to prove its optimality. We separate the cost into two groups:
Routing and BST adjustment cost: For analytical reasons, we consider a symmetric version of $\sigma^{(1)}$, named $\bar{\sigma}^{(1)}$. $\bar{\sigma}^{(1)}$ keeps the total number of requests between each pair the same but divides them half-half: for each (directed) request (u, v), we consider that half of the requests went the other direction, (v, u), making the number of requests for a pair u, v equal in both directions. This makes the frequency matrix (a matrix representation of the pair's frequencies in the demand) of $\bar{\sigma}^{(1)}$ symmetric. Theorem 6.1, that we prove later, states that

$$\bar{H} = H(\hat{Y}_{\bar{\sigma}^{(1)}} \mid \hat{X}_{\bar{\sigma}^{(1)}}) = H(\hat{X}_{\bar{\sigma}^{(1)}} \mid \hat{Y}_{\bar{\sigma}^{(1)}}) \leq H^{(1)}_{\text{con}} + 1,$$

so proving that *ReNet*'s cost on $\bar{\sigma}^{(1)}$ is $O(\bar{H})$ will prove that *ReNet*'s cost on $\sigma^{(1)}$ is $O(H^{(1)}_{\text{con}})$.

We consider the cost by node type. For a *small* node, if it connects to another *small* node, then the two nodes have a direct connection that starts from the first request and stays active for the whole $\sigma^{(1)}$ (unless the node becomes *large*, which we address later). The amortized cost for such a request is one. If a *small node* connects to a *large* node, we charge the cost for routing and the cost for adjusting the network to the large node, which we discuss now. Each *large* node w maintains a *ego-BST*(w) for its communication partners. Since *ego-BST*(w) is assumed to be a statically optimal datastructure, in our case splay trees [21], on all requests for which w is the source or destination (recall that since $\bar{\sigma}^{(1)}$ is symmetric, the frequency distribution of destinations *from* w, Y_w, and sources *to* w, X_w, are the same), it follows that the cost of these requests (routing plus adjustments) is $O(H(Y_w)) = O(H(X_w))$ (see Lemma 6.2). This cost includes all *ego-BST*(w).forward(), *ego-BST*(w).insert() and *ego-BST*(w).adjust() operations. Since each routing request involves at most one forwarding operation by a helping node between two trees (for large-large edges), the (amortized) cost of routing and tree adjustment is at most $H(\hat{Y}_{\bar{\sigma}^{(1)}} \mid \hat{X}_{\bar{\sigma}^{(1)}}) + H(\hat{X}_{\bar{\sigma}^{(1)}} \mid \hat{Y}_{\bar{\sigma}^{(1)}})$, as required.
Coordinator messages cost: We discuss the coordinator functions one-by-one:

***reset()*:** Happens once during the window $\sigma^{(1)}$. The cost is n, to broadcast the reset message to all nodes on the fixed network (using a broadcast tree).

***makeLarge()*:** Happens at most once to each node during the window. When *makeLarge*() is executed at node u, first, we are guaranteed to have enough helper nodes if needed (since the network is not full yet, see Lemma 6.3). Second, the cost is constant since we do not add new edges to u, we only replace a constant number of existing edges (u, v) (direct or via *ego-BST*(v)), with a new connection via the newly created *ego-BST*(u) of constant size. In each call of *makeLarge*, *ego-BST*(u).insert() is amortized (accounting for the adjustment cost above). The only additional cost is to notify helpers, but the number of helpers is bounded by cn and sending a message is at most $O(D)$, so the total cost is $O(cnD)$.

***addRoute()*:** Happens exactly cn times during the window $\sigma^{(1)}$. The cost of *ego-BST*(u).insert() and/or *ego-BST*(v).insert() are amortized. The only cost is to notify the helper node which is at most $O(D)$. The cost during the window is therefore $O(cnD)$.

Summing up the total cost of the coordinator messages gives $O(cnD)$. Since the number of requests in the window is $\delta = \Omega(cnD)$, the amorized cost per coordinator request is $O(c')$, for a constant c'. To this we need to add for each *ego-BST*(w) its amortized cost for routing and adjusting, but this as mention is proportional to $O(H(Y_w)) = O(H(X_w))$. Therefore the total amorized cost for the window (including routing, adjusting and coordinator messages) is $O(H^{(1)}_{\text{con}})$. ∎

To conclude the proof of Theorem 5.2, we divide σ into subsequences $\sigma^{(i)}$, separated by reset() operations. Let

$$H_{\text{con}} = \max(H(\hat{Y}_{\sigma} \mid \hat{X}_{\sigma}), H(\hat{X}_{\sigma} \mid \hat{Y}_{\sigma}))$$

and

$$H^{(i)}_{\text{con}} = \max(H(\hat{Y}_{\sigma^{(i)}} \mid \hat{X}_{\sigma^{(i)}}), H(\hat{X}_{\sigma^{(i)}} \mid \hat{Y}_{\sigma^{(i)}}))$$

A lower bound for $\text{STAT}(\sigma)$ is:

$$\text{Cost}(\text{STAT}, N^*, \sigma) \geq \Omega(H_{\text{con}}) \geq \frac{1}{k} \sum_{i=1}^{k} \Omega(H^{(i)}_{\text{con}})$$

While the cost for *ReNet* is:

$$\text{Cost}(\textit{ReNet}, \emptyset, \sigma) \leq \frac{1}{k} \sum_{i=1}^{k} O(H^{(i)}_{\text{con}}) \leq O(H_{\text{con}})$$

which makes *ReNet* statically optimal. ∎

Note that the coordination cost of the last subsequence (which may be shorter than δ), can be amortized by the coordination cost of $\sigma^{(1)}$ (which must exist since $\sigma \geq \delta$), so the amortized cost is also constant.

Observe that the cost of *ReNet* could be much lower than the cost of STAT, since STAT is also lower bounded by the conditional entropy of the whole demand σ, and not only the sum of entropies of the windows.

THEOREM 5.3. *The amortized cost of* ReNet *can be up to* $\log n$ *times lower than the cost of* STAT.

Proof. Consider for example a demand σ that is the concatenation of n demand subsequences $\sigma^{(1)}, \sigma^{(2)}, \ldots, \sigma^{(n)}$. Each demand $\sigma^{(i)}$ is of length $\Theta(n \log n)$, is sparse and has a demand graph which is a (different) two-dimensional grid. Therefore the amortized cost of *ReNet* for each $\sigma^{(i)}$ is *constant*. But if the $\sigma^{(i)}$ is *different* each time (e.g., selected round robin), then σ could be made to be uniform, where overall each source communicates to all destinations with equal frequency (over the entire σ). This will force a lower bound of $H_{\text{con}} = \Omega(\log n)$, the entropy of the uniform distribution, for STAT,. ∎

6 Deferred Proofs

6.1 Deferred Analysis of Symmetric Matrix

Let $M = M(\sigma)$ be a joint (non-symmetric) frequency matrix resulting from σ. Let $H_M(Y \mid X)$ denote the conditional entropy of Y given X under the joint probability distribution M. By definition, $H_M(Y \mid X) = H(\hat{Y}_\sigma \mid \hat{X}_\sigma)$. Let $H^*_{\text{con}} = \max(H(\hat{Y}_\sigma \mid \hat{X}_\sigma), H(\hat{X}_\sigma \mid \hat{Y}_\sigma))$, the maximum of both possible conditional entropies. Let $\bar{M} = (M + M^T)/2$ be the symmetric version of M. The conditional entropies of the symmetric and non-symmetric distributions are related as stated in the following theorem:

THEOREM 6.1. *The conditional entropy of the symmetric matrix* $\bar{M}$ *cannot be much larger than the maximal conditional entropy of* M.

$$\bar{H} = H_{\bar{M}}(Y \mid X) = H_{\bar{M}}(X \mid Y) \leq H^*_{\text{con}} + 1 \tag{6.1}$$

The proof of the theorem mainly follows from the follwoing Lemma that is based on the concavity of entropy [34] and simple entropies algebra.

LEMMA 6.1. *Let* $\tilde{p}$ *and* $\tilde{q}$ *be two probability (frequency) distributions for the same set. Let* $H^* = \max(H(\tilde{p}), H(\tilde{q}))$. *Then*

$$\frac{1}{2}H^* \leq \frac{1}{2}H(\tilde{p}) + \frac{1}{2}H(\tilde{q}) \leq H(\frac{\tilde{p}+\tilde{q}}{2}) \leq H^* + 1 \tag{6.2}$$

6.2 Other Deferred Lemmas and Proofs

LEMMA 6.2. *Consider a node* u *connected directly to the root of a statically optimal self-adjusting* ego-BST(u), *serving only requests to and from* u. *If* $\bar{p}$ *is the empirical frequency distribution of destinations and* $\bar{p}$ *is also the empirical frequency distribution of sources, then the amortized cost of routing and adjusting* ego-BST(u) *is* $O(H(\bar{p}))$.

Proof. A self-adjusting *ego-BST*(u) is originally designed to serve requests *from* the root *to* internal nodes. If the empirical frequency distribution on destinations (searched items) is $\bar{p'}$, then the amortized cost of *ego-BST*(u) is known to be $O(H(\bar{p'})$, which is optimal [21]. In our case, we also have routes from internal nodes in the tree toward the root. But for the self-adjusting *ego-BST*(u), it does not matter if the request is (u, v) or (v, u): the adjustments are the same (i.e., splay to root), hence we can assume that each route request (v, u) is actually a (u, v) request, i.e., all requests are from the root of the tree. The new empirical frequency distribution on destinations (when all requests are from root to destinations) is also $\bar{p}$. Therefore the results holds. ∎

LEMMA 6.3. (HELPING NODES) *As long as the coordinator did not call* reset(), *the size of the forwarding table of small nodes is at most* $\Delta = 6\theta$ *and helping nodes are available if needed.*

Proof. Only small nodes can be helper nodes. A small node has a maximum degree of θ, so it may need at most $12c = 3\theta$ ports in its forwarding table for the working set. For how many edges can a helper node be used as a relay? Since the number of helper nodes is at least $n/2$ (otherwise more than half of the nodes have degree larger than twice the average degree, which leads to a contradiction) and since there are at most cn large-large edges, each helper node needs to help at most $2c$ such edges. Each helper node requires 6 ports (3 for each tree), so in total it needs at most 3θ ports for helpers. Since the size of the forwarding table is 6θ, there will always be a helper node while the number of edges is less than cn which mean total size of all working sets is less than $2cn = \theta n/2$ ∎

7 Related Work

The design of effective and efficient (also in terms of cost and cabling) datacenter networks has received much interest over the last years [29, 30, 35, 36, 37, 38, 39, 40, 41, 42, 43]. The situation has been compared to the early 1980s, when many new interconnection network designs were proposed [44], not for datacenters, but for parallel computers.

The advent of technologies for reconfigurable (a.k.a. malleable [18]) networks recently introduced a new degree of freedom to the datacenter network design problem [1, 7, 10, 11, 13, 15, 16, 17, 18, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54]. By relying on movable antennas [51], mirrors [13, 50], and "disco-balls" [1], novel technologies in the context of optical circuit switches [10, 11, 12, 49], 60 GHz wireless communication [50, 55], free-space optics [1, 13], provide unprecedented topological flexibilities, allowing to quickly adapt the topology to traffic demands. Some empirical evaluations show that for certain workloads, a demand-aware network can achieve a performance similar to a demand-oblivious full-bisection bandwidth network at 25-40% lower cost [1, 13]. Empirical studies also confirm that communication patterns are often *sparse* and of *low entropy* [1, 2, 3, 4, 5], which can be exploited in demand-aware networks: in [1], it is shown that a high percentage of rack pairs does not exchange any traffic at all, while less than 1% of them account for 80% of the total traffic. In general, most bytes are delivered by large flows [40, 56, 57]. Recent surveys [6, 9] provide an overview of the literature on reconfigurable datacenter networks.

In contrast to our approach, most existing self-adjusting demand-aware network designs rely on some estimate or snapshot of the traffic demands, from which an optimized network topology is (re)computed periodically (often using exact algorithms or heuristics) [18, 30, 45, 46, 58]. This can lead to unnecessary reconfigurations as the optimal solutions for two different snapshots are very sensitive to changes in the input and can look completely different, although a similar outcome could have been achieves with small reconfigurations only. Furthermore, periodic reconfigurations introduce the problem that they are either too coarse-grained or too fine-grained, and do not fully exploit the potential of demand-awareness. We in this paper present a more refined and adaptive model, accounting also for the reconfiguration costs, and allowing us to study (within our model) the tradeoff between the costs and benefits of reconfigurations. Moreover, in contrast to most existing algorithms relying on mixed integer programming, our algorithms are efficient (*polynomial-time*), and in contrast to existing algorithms relying on heuristics, our approach comes with *provable* guarantees, even over time.

In terms of formal guarantees, an upper bound on what can be achieved in terms of statically optimized demand-aware networks is due to Avin et al. [7], who build upon initial insights derived in [47, 59]. We in this paper leverage the degree reduction technique of [7], however, to derive a very different result. The fixed demand-aware network designs by Avin et al. [7] have recently also been extended to optimize for load, in addition to route lengths [8]. SplayNets [47, 59] also rely on splay trees to adjust the network, and dynamically adapt to changing traffic patterns. However, besides their convergence properties under specific fixed demands, these networks do not provide any optimality but only heuristic guarantees. In fact, this is an inherent limitation, as static optimality is impossible to achieve based on single tree networks (unless the demand has a specific structure, e.g., comes from a single node, in which case dynamic optimality can be achieved using networks based on *push-down trees* [60]). Indeed, to the best of our knowledge, so far, no result existed on how to actually match the lower bound provided in [47], without perfect knowledge of the demand. Recently, Bienkowski et al. [47] considered the problem of how to schedule b reconfigurable optical switches in a hybrid datacenter, to minimize the expected path length over time; the reconfiguration cost is α per link, where α is a parameter. The authors showed that this problem can be seen as an online b-matching problem and presented a constant competitive algorithm, exploiting a connection to online paging. In their setting, however, routing is strictly segregated [46]: if an optical link is used, it needs to connect the source and destination directly.

Our model also features interesting connections to switch scheduling [61, 62]: In classic switch scheduling, packets arriving at a switch need to be moved from the input buffer to the output buffer, and in each time step, the input buffers and all their output buffers must form a bipartite matching. A striking result of Chuang, Goel, McKeown, and Prabhakar showed that a switch using input/output queuing with a speed-up of 2 can simulate a switch that uses pure output queuing [62]. This connection has recently been exploited by Venkatakrishnan et al. to design offline scheduling algorithms for reconfigurable datacenters [53]. The authors consider a setting in which demand matrix entries are small, and analyze a greedy algorithm achieving an (almost) tight approximation guarantee. In particular, their model allows to account for reconfiguration delays, which are not captured by traditional crossbar switch scheduling algorithms, e.g., relying on centralized Birkhoff-von- Neumann decomposition schedulers [63]. Schwartz et al. [64] recently presented online greedy algorithms for this problem, achieving a provable competitive ratio over time. A powerful approach to design online algorithms is based on primal-dual techniques, also leveraged by Dinitz et al. [17] to optimize flow and completion times: in [17], a model is considered where the demands are the edges in an arbitrary graph, and in each round, a vertex cover can be communicated: each node can only send certain number of packets in one round. Kulkarni et al. [65]

generalize classic online packet scheduling to a two-tiered reconfigurable datacenter network, as it is for example used by ProjecToR [1]; using a primal-dual analysis of a stable matching algorithm the authors show a competitive ratio of $2\cdot(2/\epsilon+1)$ in a resource augmentation model: the online algorithm runs $2+\epsilon$ times faster. These lines of research however are technically fairly different from ours: they either consider a maximization problem, aiming to maximize the total data transmission for a certain time window, or to minimize flow times, whereas in our model, we aim to minimize routing and reconfiguration costs.

We are not aware of any demand-aware network which provides compact and local routing, or arbitrary addressing. We believe that this aspect of our work is of independent interest, as it shows, for the first time, that highly scalable reconfigurable networks are possible at least in theory, and can be a stepping stone toward a better understanding of the important cross-layer issues arising in reconfigurable networks.

8 Conclusion

This paper presented the first self-adjusting network which provides entropy-proportional (and hence statically optimal) route lengths while minimizing reconfigurations costs and accounting for scalability, through compact and local routing. Our approach leveraged an intriguing connection to self-adjusting datastructures, using self-adjusting BSTs as building blocks (i.e., per-source "ego-trees"), and combining them to a *network*.

We believe that our work opens several interesting directions for future research. First, while this paper shows that scalable demand-aware networks achieving all the desirable properties above are feasible *in theory*, further refinements are needed in order to tailor our approach to practical use cases or specific objective functions such as flow completion times. It also remains to further explore the implications of our approach (and of reconfigurable networks in general) on other layers, such as routing and congestion control, as well as on scalability. An intriguing open theoretical question from our work regards the design of self-adjusting DANs which optimize metrics related to *temporal* locality, such as dynamic optimality in specific settings. More generally, we believe that self-adjusting networks can be of interest beyond the datacenter context considered here.

References

[1] 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 *Proc. ACM SIGCOMM*, (New York, NY, USA), pp. 216–229, ACM, 2016.

[2] C. Avin, M. Ghobadi, C. Griner, and S. Schmid, "On the complexity of traffic traces and implications," in *Proc. ACM SIGMETRICS*, 2020.

[3] A. Roy, H. Zeng, J. Bagga, G. Porter, and A. C. Snoeren, "Inside the social network's (datacenter) network," in *Proc. ACM SIGCOMM Computer Communication Review (CCR)*, vol. 45, pp. 123–137, ACM, 2015.

[4] S. Kandula, S. Sengupta, A. Greenberg, P. Patel, and R. Chaiken, "The nature of data center traffic: measurements & analysis," in *Proc. 9th ACM Internet Measurement Conference (IMC)*, pp. 202–208, 2009.

[5] 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.

[6] K.-T. Foerster and S. Schmid, "Survey of reconfigurable data center networks: Enablers, algorithms, complexity," in *SIGACT News*, 2019.

[7] C. Avin, K. Mondal, and S. Schmid, "Demand-aware network designs of bounded degree," in *Proc. International Symposium on Distributed Computing (DISC)*, 2017.

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

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

[10] 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," *Proc. ACM SIGCOMM Computer Communication Review (CCR)*, vol. 40, no. 4, pp. 339–350, 2010.

[11] 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 *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, pp. 267–280, ACM, 2017.

[12] G. P. R. S. N. Farrington, A. F. P. Chen-Sun, T. R. Y. F. G. Papen, and A. Vahdat, "Integrating microsecond circuit switching into the data center," 2013.

[13] 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 *Proc. ACM SIGCOMM Computer Communication Review (CCR)*, vol. 44, pp. 319–330, 2014.

[14] 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 *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*, pp. 782–797, 2020.

[15] 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.

[16] 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," *arXiv preprint arXiv:1903.12307*, 2019.

[17] M. Dinitz and B. Moseley, "Scheduling for weighted flow and completion times in reconfigurable networks," *Proc. IEEE INFOCOM*, 2020.

[18] A. Singla, A. Singh, K. Ramachandran, L. Xu, and Y. Zhang, "Proteus: a topology malleable data center network," in *Proc. ACM Workshop on Hot Topics in Networks (HotNets)*, 2010.

[19] M. Wang, Y. Cui, S. Xiao, X. Wang, D. Yang, K. Chen, and J. Zhu, "Neural network meets dcn: Traffic-driven topology adaptation with deep learning," *Proc. ACM Meas. Anal. Comput. Syst.*, vol. 2, pp. 26:1–26:25, June 2018.

[20] J. S. Vitter, "Design and analysis of dynamic Huffman codes," *Journal of the ACM*, vol. 34, pp. 825–845, oct 1987.

[21] D. Sleator and R. Tarjan, "Self-adjusting binary search trees," *Journal of the ACM (JACM)*, vol. 32, no. 3, pp. 652–686, 1985.

[22] P. Auer, N. Cesa-Bianchi, Y. Freund, and R. E. Schapire, "The nonstochastic multiarmed bandit problem," *SIAM journal on computing*, vol. 32, no. 1, pp. 48–77, 2002.

[23] Y. Freund and R. E. Schapire, "Adaptive game playing using multiplicative weights," *Games and Economic Behavior*, vol. 29, no. 1-2, pp. 79–103, 1999.

[24] N. Littlestone and M. K. Warmuth, "The weighted majority algorithm," *Information and computation*, vol. 108, no. 2, pp. 212–261, 1994.

[25] N. Bansal, A. Blum, S. Chawla, and A. Meyerson, "Online oblivious routing," in *Proc. ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pp. 44–49, 2003.

[26] E. D. Demaine, D. Harmon, J. Iacono, and M. Patrascu, "Dynamic optimality - almost," *SIAM J. Comput.*, vol. 37, no. 1, pp. 240–251, 2007.

[27] A. Borodin and R. El-Yaniv, *Online computation and competitive analysis*. cambridge university press, 2005.

[28] S. Hoory, N. Linial, and A. Wigderson, "Expander graphs and their applications," *ulletin of the American Mathematical Society*, vol. 43, 2006.

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

[30] A. Singla, "Fat-free topologies," in *Proc. 15th ACM Workshop on Hot Topics in Networks (HotNets)*, pp. 64–70, 2016.

[31] G. N. Frederickson and R. Janardan, "Designing networks with compact routing tables," *Algorithmica*, vol. 3, no. 1, pp. 171–190, 1988.

[32] M. Thorup and U. Zwick, "Compact routing schemes," in *Proc. ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, 2001.

[33] A. Fiat, R. M. Karp, M. Luby, L. A. McGeoch, D. D. Sleator, and N. E. Young, "Competitive paging algorithms," *Journal of Algorithms*, vol. 12, no. 4, pp. 685–699, 1991.

[34] T. M. Cover and J. A. Thomas, *Elements of information theory*. John Wiley & Sons, 2012.

[35] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," in *Proc. ACM SIGCOMM Computer Communication Review (CCR)*, vol. 38, pp. 63–74, 2008.

[36] C. Guo, H. Wu, K. Tan, L. Shi, Y. Zhang, and S. Lu, "Dcell: a scalable and fault-tolerant network structure for data centers," in *Proc. ACM SIGCOMM Computer Communication Review (CCR)*, vol. 38, pp. 75–86, 2008.

[37] 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," *Proc. ACM SIGCOMM Computer Communication Review (CCR)*, vol. 39, no. 4, pp. 63–74, 2009.

[38] A. Singla, C.-Y. Hong, L. Popa, and P. B. Godfrey, "Jellyfish: Networking data centers, randomly.," in *Proc. USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, vol. 12, pp. 17–17, 2012.

[39] H. Wu, G. Lu, D. Li, C. Guo, and Y. Zhang, "Mdcube: a high performance network structure for modular data center interconnection," in *Proc. ACM International Conference on Emerging Networking Experiments and Technologies (CONEXT)*, pp. 25–36, 2009.

[40] A. 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 *Proc. ACM SIGCOMM Computer Communication Review (CCR)*, vol. 39, pp. 51–62, 2009.

[41] V. Liu, D. Halperin, A. Krishnamurthy, and T. Anderson, "F10: A fault-tolerant engineered network," in *Proc. USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, nsdi'13, (Berkeley, CA, USA), pp. 399–412, USENIX Association, 2013.

[42] M. Besta and T. Hoefler, "Slim fly: A cost effective low-diameter network topology," in *Proc. International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, pp. 348–359, 2014.

[43] A. Singla, P. B. Godfrey, and A. Kolla, "High throughput data center topology design.," in *Proc. USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, pp. 29–41, 2014.

[44] A. Akella, T. Benson, B. Chandrasekaran, C. Huang, B. Maggs, and D. Maltz, "A universal approach to data center network design," in *Proc. International Conference on Distributed Computing and Networking*

(ICDCN), p. 41, ACM, 2015.

[45] K.-T. Foerster, M. Ghobadi, and S. Schmid, "Characterizing the algorithmic complexity of reconfigurable data center architectures," in *Proc. ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*, 2018.

[46] T. Fenz, K.-T. Foerster, S. Schmid, and A. Villedieu, "Efficient non-segregated routing for reconfigurable demand-aware networks," in *Proc. IFIP Networking*, 2019.

[47] 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, pp. 1421–1433, June 2016.

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

[49] 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 *Proc. USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, vol. 14, pp. 1–15, 2014.

[50] 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," *Proc. ACM SIGCOMM Computer Communication Review (CCR)*, vol. 42, no. 4, pp. 443–454, 2012.

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

[52] 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.

[53] S. Bojja Venkatakrishnan, M. Alizadeh, and P. Viswanath, "Costly circuits, submodular schedules and approximate carathéodory theorems," in *Proceedings of the 2016 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Science*, pp. 75–88, 2016.

[54] M. Wang, Y. Cui, S. Xiao, X. Wang, D. Yang, K. Chen, and J. Zhu, "Neural network meets dcn: Traffic-driven topology adaptation with deep learning," *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, vol. 2, no. 2, p. 26, 2018.

[55] S. Kandula, J. Padhye, and P. Bahl, "Flyways to decongest data center networks," 2009.

[56] G. Judd, "Attaining the promise and avoiding the pitfalls of tcp in the datacenter.," in *Proc. USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, pp. 145–157, 2015.

[57] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan, "Data center tcp (dctcp)," in *Proc. ACM SIGCOMM Computer Communication Review (CCR)*, vol. 40, pp. 63–74, 2010.

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

[59] B. Peres, O. Souza, O. Goussevskaia, S. Schmid, and C. Avin, "Distributed self-adjusting tree networks," in *Proc. IEE INFOCOM*, 2019.

[60] C. Avin, K. Mondal, and S. Schmid, "Dynamically optimal self-adjusting single-source tree networks," in *Proc. Latin American Theoretical Informatics Symposium (LATIN)*, 2020.

[61] N. McKeown, "The islip scheduling algorithm for input-queued switches," *IEEE/ACM transactions on networking*, vol. 7, no. 2, pp. 188–201, 1999.

[62] S.-T. Chuang, A. Goel, N. McKeown, and B. Prabhakar, "Matching output queueing with a combined input/output-queued switch," *IEEE Journal on Selected Areas in Communications*, vol. 17, no. 6, pp. 1030–1039, 1999.

[63] N. McKeown, A. Mekkittikul, V. Anantharam, and J. Walrand, "Achieving 100% throughput in an input-queued switch," *IEEE Transactions on Communications*, vol. 47, no. 8, pp. 1260–1267, 1999.

[64] R. Schwartz, M. Singh, and S. Yazdanbod, "Online and offline greedy algorithms for routing with switching costs," *arXiv preprint arXiv:1905.02800*, 2019.

[65] J. Kulkarni, S. Schmid, and P. Schmidt, "Scheduling opportunistic links in two-tiered reconfigurable datacenters," in *ArXiv Technical Report*, 2020.