CBNet: Demand-Aware Tree Topologies for Reconfigurable Datacenter Networks

Otavio A. de O. Souza[a], Olga Goussevskaia[a], Stefan Schmid[b]

[a] *Computer Science Department, Universidade Federal de Minas Gerais (UFMG), Rua Reitor Pires Albuquerque, Belo Horizonte, 31270-901, Minas Gerais, Brazil*
[b] *Computer Science Department, University of Vienna, Waehringer Strasse 29, Vienna, 1090, Austria*

Abstract

This paper studies the design of demand-aware network topologies: networks that dynamically adapt themselves toward the demand they currently serve, in an online manner. While demand-aware networks may be significantly more efficient than demand-oblivious networks, frequent adjustments are still costly. Furthermore, a centralized controller of such networks may become a bottleneck.

We present CBNet (Counting-Based self-adjusting Network), a demand-aware network that relies on a distributed control plane supporting concurrent adjustments, while significantly reducing the number of reconfigurations, compared to related work. CBNet comes with formal guarantees and is based on concepts of self-adjusting data structures. We evaluate CBNet analytically and empirically and we find that CBNet can effectively exploit locality structure in the traffic demand.

Keywords: Self-adjusting networks, concurrency, online algorithms

1. Introduction

Empirical studies show that communication patterns in datacenters feature much spatial and temporal structure [1, 2, 3, 4], i.e., traffic is bursty, and traffic matrices skewed. This structure represents an untapped potential for building more efficient communication networks: today's datacenter networks are designed in a manner which is entirely *oblivious* to the communication pattern.

In contrast, a demand-aware network, e.g., based on emerging reconfigurable optical technologies, may self-adjust to better serve the elephant flows in the network [5, 6, 2, 7, 8, 9, 10].

A key challenge in the design of self-adjusting networks is to strike a balance between the benefits and costs of reconfigurations: while reconfigurations allow to reduce communication costs, by moving frequently communicating nodes (e.g., top-of-rack switches) topologically closer, such reconfigurations should be used in moderation: reconfigurations take time and may temporarily lead to packet loss. A second challenge is related to the collection of data about the demand and the decision making based on this data: these are two inherent operations in any demand-aware network, but may become an operational bottleneck. Furthermore, communication patterns may not be perfectly predictable, and hence, reconfiguration decisions need to be taken *in an online manner*.

This paper investigates the algorithmic problem underlying such demand-aware and self-adjusting networks. In particular, to address the above challenges, we introduce CBNet, a distributed and concurrent demand-aware network based on tree topologies, which aims to reduce reconfigurations compared to the state-of-the-art solutions, such as SplayNet [7] and DiSplayNet [11, 12].

CBNet is based on concepts from self-adjusting data structures, and in particular, CBTrees [13]. CBNet gradually adapts the network topology toward the communication pattern in an online manner, i.e., without previous knowledge of the demand distribution. At the same time, *bidirectional semi-splaying* and counters are used to maintain state, minimize reconfiguration costs and maximize concurrency.

Contributions. Our main contribution is CBNet, a counting-based self-adjusting network, which adapts to the unknown traffic pattern in a demand-aware and online manner, while minimizing reconfiguration costs. CBNet relies on decentralized and concurrent algorithms and comes with provable (worst-case) performance guarantees, and also fares well under realistic workloads: we report on extensive simulations using both real and synthetic workloads, and find that CBNet can indeed outperform state-of-the-art, by better leveraging

locality and reducing costs.

Paper organization. In Section 2 we discuss related work. In Section 3 we present the model and problem definitions. In Section 4 we present an overview of CBNet. In Section 5 we describe the concept of counting-based reconfiguration. In Section 6 we present the sequential version of CBNet algorithm, and in Section 7 we analyze its performance. In Section 8 we present and analyze the concurrent version of CBNet. In Section 9 we describe the workloads used in our experiments and analyze them in terms of temporal and non-temporal locality. In Section 10 we report on simulations, and in Section 11 present our conclusions.

2. Related Work

Traditionally, network designs rely on static topologies, such as the Clos topology [14, 15, 16], hypercubic topologies like BCube and MDCube [17, 18], or expander-based networks [19, 20] in datacenters.

Recently, reconfigurable topologies have received much attention, which come in two flavors, demand-oblivious e.g. [21, 22, 23] and demand-aware, e.g. [2, 24, 25, 26, 7, 27, 11, 28, 8, 29, 30, 31], or a combination of both, e.g., Cerberus [32]. Most existing demand-aware architectures rely on an estimation of traffic matrices [33, 28, 8, 29, 34, 35] which can limit the granularity and reactivity of the network, but there are also more fine-grained approaches such as [25, 26], which however rely on a centralized control.

We in this paper are interested in distributed solutions, such as ProjecToR [2] or DiSplayNet [12] supporting fine-grained reconfiguration. The most closely related papers to our work are [7, 12, 13], which also rely on concepts from self-adjusting data structures (see [6] for an overview of this approach). Note that, in contrast to CBNet, [7] is still centralized, [12] comes with significant adjustment costs. CBNet extends the concept of a CBTree [13], defined as a counting-based self-adjusting binary search tree in which, as in splay trees [36], more-frequently accessed items move closer to the root. Unlike the original splay tree, in which

each access moves the accessed item all the way to the root via a sequence of rotations, accesses in a CBTree do just $O(n + n \log(m/n)$ rotations during a sequence of n operations.

A possible application for CBNet is Reconfigurable Datacenter Networks (RDCNs). The majority of programmable RDCNs with on-demand reconfigurations use full crossbar, 3D-MEMS-based optical circuit switches (OCS) [37, 38], or Wavelength Division Multiplexing-based switching [39]. While promising performance results have been demonstrated with various prototypes of demand-aware reconfigurable networks, today, it is often challenging to experiment with these technologies, as they are usually based on custom-built prototypes and rely on tailored hardware and software which is not publicly available. One example of a framework that supports experimentation and reproducibility is ExReC [40]. It uses off-the-shelf hardware (such as Polatis Series 6000n 32×32 OCS [41]) for evaluating different hybrid reconfigurable topologies and applications, such as distributed machine learning training.

3. Model

In this work the objective is to design and formally analyze the performance of distributed algorithms for self-adjusting networks. The network should connect a set V of n communication nodes (e.g., top-of-rack switches or peers). The input to the problem is given by a sequence σ of m messages $\sigma_i(s_i, d_i) \in V \times V$ occurring over time, with source s and destination d; m can be infinite. We denote by b_i the time when a message σ_i is generated, and by e_i the time in which it is delivered. The time between successive requests arrivals is assumed to be at least one. The sequence σ is revealed over time, in an online manner: the algorithm does not have any information about the future requests σ_j at time $t < b_j$. Moreover, the sequence σ can be arbitrary: in our formal analysis we consider a worst-case scenario, where σ is chosen *adversarially*, as to maximize the cost of a given algorithm.

We will focus on networks based on *Binary Search Trees* (BST), because trees

are a basic graph family and we envision that the self-adjusting links constitute only a subset of the topology, a usual assumption in such networks [2]. Moreover, BSTs are *locally routable*, i.e., dynamic topological changes do not require the global recomputation of routes. We denote the family of BST networks by $\mathcal{T} = T_0, T_1, \ldots$, where T_0 is an arbitrary BST.

Distributed reconfiguration. In order to minimize the communication cost and adjust the topology smoothly over time, the tree is reconfigured locally through rotations that preserve the BST properties. One rotation updates a constant number of links at constant cost. Accordingly, we will denote the tree at time (round) t computed by a given distributed algorithm (possibly accounting for the communication requests $\sigma_{t'}$ with $t' < t$) by $T_t \in \mathcal{T}$. From now on, we use the terms *rotation* and (local network) *reconfiguration* interchangeably to refer to local topological updates in the tree.

Bidirectional semi-splaying. Differently from SplayNet [7] and DiSplayNet [11], where the classical *zig, zig-zig* and *zig-zag* splay operations have been employed exclusively in a bottom-up direction, CBNet leverages *bottom-up* and *top-down* semi-splaying (*semi-zigzig* and *semi-zigzag*) operations, first introduced for splay trees [36] and later adapted for top-down communication in CBTrees [13]. Besides being simpler to implement in a distributed setting, the semi-splay operations have a lower communication cost than splaying. Note that (semi) splaying not only moves a node upwards in the tree, preserving BST properties, but also roughly halves the depth of every node along the communication path. This halving effect makes splaying efficient in an amortized sense and is a property not shared by other, simpler rotation heuristics, such as move-to-root [42].

Time model. We assume that time is divided into synchronous time slots, in which a message can travel a constant number of hops in the network or a local reconfiguration might be performed. In order to study concurrency, we divide the execution time in *rounds*. Each round is comprised of a constant number of time slots, in which a local consensus can be reached and one reconfiguration completed. In a round, multiple (independent) nodes can make local

reconfigurations (steps) concurrently. We consider that nodes and communication between them are reliable and synchronous. In terms of time, we aim to minimize the makespan:

Definition 1. ***Time cost:*** *Consider any initial binary tree T_0, a sequence of m messages σ and algorithm $\mathcal{A}$. $Makespan(\mathcal{A}, T_0, \sigma) = \max_{1 \leq i \leq m} e_i - b_1$.*

3.1. Refined cost model

In CBNet, as messages travel from the source to the destination, rotations are traded with routing operations. Therefore, we distinguish between the work needed to forward a message and the work needed to perform local reconfigurations. In practice, the cost of performing a network topology reconfiguration is typically higher than that of forwarding a message over a communication link. We assume that routing a message incurs a cost of 1 unit per hop and that a rotation incurs a cost of $R = O(1)$.

Consider a sequence σ of m messages, an algorithm $\mathcal{A}$, a *BST* T_0, and a message $\sigma_i(s_i, d_i) \in \sigma$. Let us define the reconfiguration cost ω_i as the number of rotations performed by $\mathcal{A}$ to deliver σ_i. The routing cost will be equal to $l_{e_i}(s_i, d_i)$, the length of the path $\mathcal{P}_{e_i}(s_i, d_i)$ in the resulting tree T_{e_i}, i.e., after σ_i has been delivered. Note that $l_{e_i}(s_i, d_i)$ is not necessarily one, as in [7, 11], but is equal to the number of times the message was forwarded along $\mathcal{P}_{b_i}(s_i, d_i)$, instead of triggering a rotation. We assume that the value of the routing cost to deliver a message, even when it is addressed to itself ($\sigma_i(s_i, s_i)$), is ≥ 1.

Definition 2. ***Work cost:*** *Consider any initial binary tree T_0, a sequence of m messages $\sigma_i(s_i, d_i) \in \sigma$ and algorithm $\mathcal{A}$. We define the* ***total routing cost****,* ***total reconfiguration cost****, and* ***total work cost****, respectively, as follows:*

$$\begin{aligned} \mathcal{L}(\mathcal{A}, T_0, \sigma) &= \sum_{i=1}^{m} (l_{e_i}(s_i, d_i) + 1), \\ \Omega(\mathcal{A}, T_0, \sigma) &= R \times \sum_{i=1}^{m} \omega_i, \\ \mathcal{C}(\mathcal{A}, T_0, \sigma) &= \mathcal{L}(\mathcal{A}, T_0, \sigma) + \Omega(\mathcal{A}, T_0, \sigma). \end{aligned}$$

3.2. Amortized analysis and the potential method

We are interested in the worst-case performance over arbitrary sequences of operations (rather than individual operations), and hence, conduct an *amortized analysis*.

The potential method defines a function that maps a data structure onto a real-valued, non-negative "potential". The potential stored in the data structure may be used to pay for future operations. The potential of the tree at time i (i.e., the potential of T_i) is represented as $\phi(T_i)$. In the potential method, the amortized cost $\hat{c}_i$ of an operation i is the actual cost c_i plus the increase in potential δ due to the operation, where $\delta = \phi(T_i) - \phi(T_{i-1})$. This gives us: $\hat{c}_i = c_i + \phi(T_i) - \phi(T_{i-1})$.

The amortized cost and potential function must be defined in order to always maintain such equivalence. This implies that if the actual cost of and operation is less than the amortized cost, the potential is increased, and if the cost of an operation is greater than its amortized cost, the potential is decreased. Using the definition of $\hat{c}_i$, we can derive the total amortized cost given the actual costs: $\sum_{i=1}^{n} \hat{c}_i = \sum_{i=1}^{n} c_i + \phi(T_n) - \phi(T_0)$.

Every time when performing an operation, the node can: (1) pay the actual work; (2) pay more than the actual work, such that the excess is deposit in an "account", i.e., increase the potential; (3) pay less than the actual work, making withdrawals from the account to cover the deficiency, i.e., decrease the potential. Moreover, an *invariant* is maintained: the total number of credits in the data structure ≥ 0.

We define the amortized cost for CBNet as follows:

Definition 3. ***Amortized cost:*** *Given a sequence of m messages σ, if $\mathcal{C}(\sigma_i)$ is the (time or work) cost of the $\sigma_i \in \sigma$, the amortized cost is defined with respect to the worst sequence σ and initial tree T_0 as: $\mathcal{C}_A = \max_{\sigma, T_0} \frac{1}{m} \sum_{\sigma_i \in \sigma} \mathcal{C}(\sigma_i)$.*

One last useful concept for the analysis is that of empirical entropy.

Definition 4. ***Empirical entropy:*** *Given a sequence of m messages σ and initial tree T_0 on n nodes, the empirical entropy is defined with respect to $\hat{S}$ as:*

$H(\hat{S}) = \sum_{i=1}^{n} f_s(v_i) \log \frac{1}{f_s(v_i)}$, *where* $\hat{S} = \{f_s(v_1), \ldots, f_s(v_n)\}$ *are the frequencies that a node* $v_i \in V$ *is a source in* σ*. Similarly,* $H(\hat{D})$ *is defined for the set of destination frequencies* $\hat{D}$*.*

4. CBNet overview

State-of-the-art distributed self-adjusting networks, such as SplayNet [7] and DiSplayNet [12] are based on the self-adjusting binary search trees *splay trees* [36]. They gradually adapt the network topology toward the communication pattern in an online manner. These self-adjusting networks, however, face challenges when it comes to their application in practice.

Problematic inheritance from data structures. The communication model underlying SplayNet and DiSplayNet is not entirely distributed nor realistic. They adopt an aggressive reconfiguration strategy: the source and the destination nodes of each message execute a sequence of bottom-up rotations, until they meet at their lowest common ancestor (LCA), at which point they finally exchange the data. Behind this approach lie two problematic assumptions: (1) the destination node knows when a message is generated toward it and starts rotations simultaneously with the source node; and (2) the destination node travels in the network without carrying any data in order to meet the source node, so both the source and the destination nodes are locked until the message is delivered, limiting concurrency.

Less adjustments. CBNet is based on a different, concurrent self-adjusting data structure, the CBTree [13], in which rotations are traded for routing operations. CBNet performs rotations infrequently, an amortized subconstant $o(1)$ per operation, drastically reducing the reconfiguration cost, while preserving the amortized communication cost guarantees.

More realistic distributed communication. CBNet enables a more realistic and fully distributed communication model. Bidirectional semi-splaying and the ability to forward messages allow for the natural behavior of a message traveling through the network: a message moves bottom-up in the tree when it

is navigating towards the root and top-down, otherwise. To the extent of our knowledge, CBNet is the first *message-oriented* self-adjusting network.

More concurrency. Frequent network reconfigurations limit the potential for concurrent execution of self-adjusting networks because rotations might result in conflicts between concurrent communication requests. By drastically reducing the number of reconfigurations and by freeing the source and the destination nodes of each message, CBNet scales better with the concurrency level.

5. Communication history through counters

CBNet keeps track of the communication history through counters and weights, like the CBTree [13]. Each node $v \in V$ maintains a local variable $c_t(v)$ that accounts for the number of times v has been the source or the destination of some communication request: $c_t(v) = |\{\sigma_i | \sigma_i \in \sigma,\ v = src(\sigma_i) || v = dst(\sigma_i),\ e_i \leq t\}|$, and $c(v) = c_{e_m}(v), \forall v \in V$. We define **weights** as:

$$W_t(v) = \sum_{x \in T_t(v)} c_t(x), W(v) = W_{e_m}(v), \forall v \in V, \tag{1}$$

where e_m is the time of delivery of the last message $\sigma_m \in \sigma$ and $T_t(v)$ is the subtree rooted at v in time-slot $0 \leq t \leq e_m$. In our implementation of CBNet, nodes maintain only their weights, and the counter value is obtained as: $c_t(v) = W_t(v) - W_t(v.l) - W_t(v.r), \forall v \in V, t \geq 0$, where $v.l$ and $v.r$ are the left and right children of node v in $T_t(v)$, respectively.

We define the **rank** of a node as: $r_t(v) = \log W_t(v), \forall v \in V, t \geq 0$, assuming $r_t(v) = 0$ if $W_t(v) = 0$. And we define the **potential** of a network by using the potential function presented in [13] as:

$$\Phi_t = \Phi(T_t) = \sum_{v \in V} r_t(v), t \geq 0. \tag{2}$$

Local computation of potential change. The potential of the network starts with the value of zero. It increases over time due to the arrival of new messages or decreases as a result of a rotation. In order to decide whether a given message will cause a rotation or will be forwarded, the network potential

difference $\Delta\Phi_t$ that would result from that rotation has to be computed locally in time-slot t. Since the network potential is the sum of the ranks of each node in the tree, and since the ranks of all nodes that do not participate in a rotation have their ranks unchanged, $\Delta\Phi_t$ depends only on the type of rotation (top-down or bottom-up, *semi zig-zig* or *semi zig-zag*) and the rank variation of the neighboring nodes that would participate in that rotation [13]. If the rotation has been performed, the weights of the participating nodes are updated to reflect the change in network potential.

6. Sequential CBNet

When a message enters the network, it moves from the source toward the destination node by means of two kinds of operations, referred to as routing steps or rotation steps. Routing steps move the message between nodes while keeping the network's topology unchanged. Rotation steps move the message by restructuring the topology of the network. We define the *current node* of a message as the node currently holding the message at a given point in time. Below we formally define a *step*.

Definition 5. *$Step_t(\sigma_i, x_t)$: Given a message $\sigma_i(s_i, d_i) \in \sigma$ with begin and end times b_i and e_i, respectively, and a BST instance $T_t, b_i \leq t \leq e_i$, a* ***step*** *$step_t(\sigma_i(s_i, d_i), x_t)$ is an operation performed by the message's current node, $x_t \in \mathcal{P}_t(s_i, d_i)$, that reduces the message's distance to its destination d, while preserving the BST properties of the network. A sequence of steps, starting at the source node, $x_{b_i} = s_i$, deliver the message to the destination node $x_{e_i} = d_i$. Each step can move the message along the path $\mathcal{P}_t(s_i, d_i)$ in a* ***bottom-up*** *or* ***top-down*** *direction and can trigger an operation of type* ***routing*** *($forward(T_t, \sigma_i, direction)$) or of type* ***rotation*** *($splay(T_t, \sigma_i, direction, splayType)$), where $splayType \in \{semi-zigzag, semi-zigzig\}$.*

Since there is no central information accessible to nodes about the network structure, to determine the direction of a message, each node $v \in V$ stores the identifiers of its direct neighbors in the tree, i.e., its parent ($v.p$), its left child

($v.l$), its right child ($v.r$), as well as the smallest ($v.smallest$) and the largest ($v.largest$) identifiers currently present in the sub-tree rooted at v. With this information, given the destination identifier, a node x can decide if the message $\sigma_i(s_i, d_i)$ must be forwarded to its parent ($d_i \geq x.smallest$ and $d_i \leq x.largest$), its right child ($x < d_i \leq x.largest$), or left child ($x.smallest \leq d_i < x$) in the tree, where x and d_i stand for the identifiers of the respective nodes. The rotation type of the step is determined by the position of the current node relative to its neighbors and the message's direction, as defined in [13]. Whether the step is of type rotation or routing is determined by the network potential difference that a rotation would cause, as described below.

Algorithm 1: While a message $\sigma_i(s_i, d_i) \in \sigma$ travels from source to destination, each current node $x_t \in T_t, b_i \leq t \leq e_i$ executes the step routine, shown in Algorithm 1. If the $LCA_t(s_i, d_i)$ has been reached (lines $2-3$) a weight-update message is sent in the bottom-up direction, toward the root ρ of the tree. This update message is a small control message that carries no data and increments the weights of all nodes $v \in \mathcal{P}_t(LCA_t(s_i, d_i), \rho)$ by 2 (1 for s_i and 1 for d_i). Note that it does trigger rotations on its way, like a regular message; we therefore include it in the work cost analysis of CBNet. Then, the current node decides upon which type of rotation (line 4) and direction (line 5) to perform. This decision is based on the relative position of d_i and the 2-hop neighbors of the current node. It then computes the network potential difference that the latter would cause (line 6). All these computations are performed without global knowledge about the network topology information from two hops ahead in the path $\mathcal{P}_t(x_t, d_i)$ is enough, as described in Section 5.

If the rotation decreases the total potential of the network by more than a constant $\delta \in (0, 2]$ (we used $\delta = 2$ in our implementation), then a rotation is performed, updating the tree topology (lines $7-8$); otherwise, the topology remains unchanged and the message is forwarded to the next current node (line 10). Finally, the weights of the nodes that participated in this step and remained on the path from s_i to d_i after the step are incremented by 1 (line 11);

6.1. Adaptive CBNet

CBNets have the property that messages follow an expected path length proportional to the entropy of the communication distribution, which is computed based on the counting history stored by each node. Considering that the request sequence is infinite in a real network, a long history stored in the counters might act as a "legacy burden" and slow the rate of reconfigurations, making CBNet less reactive to temporal locality, especially when the traffic demand distribution goes through significant changes after a long sequence of requests [43]. In order for the topology not to become too static, a counter resetting mechanism should be implemented, so that older requests contribute less to the current weights used in the potential computations.

To address this problem, we implemented and simulated an adaptive version of CBNet, to which we refer as Adaptive CBNet. In Algorithm 1, after line 5, a call is added to function $decayWeight()$ (Algorithm 2). When a sufficiently long time ($timeOut$) elapses since the last counter update of a node ($timeStamp$), the node's weight (W_t) is decreased according to an exponential back-off scheme (line 3). A similar approach has been proposed in the context of CBTrees [13].

7. Sequential CBNet analysis

We use the potential function defined in (2) to amortize actual costs in our analysis. Consider an initial (arbitrary) BST network instance T_0 of n nodes, a message sequence σ of m requests, a message $\sigma_i(s_i, d_i) \in \sigma$ generated at time b_i and delivered at time e_i. Recall that the amortized cost of a step operation is the actual cost, plus the increase in potential caused by it: $\hat{c}_t(step_t) = \mathcal{C}_t(step_t) + \Phi(T_t) - \Phi(T_{t-1})$. Therefore, if σ_i requires of l_i steps of cost $O(1)$, the actual cost to fulfill σ_i is $\sum_{t=1}^{l_i} O(1)$, causing a potential change of $\sum_{t=1}^{l_i} \Phi(T_t) - \Phi(T_{t-1})$. This summation results in a telescoping series in which all terms cancel except the first and the last, resulting in amortized cost $\hat{c}(\sigma_i) = l_i + \Phi(T_{e_i}) - \Phi(T_{b_i})$.

We start the analysis with an auxiliary lemma, proved in [13].

Algorithm 1 Sequential CBNet $step_t(\sigma(s_i, d_i), x_t)$ executed by the current node $x_t \in V$ of message $\sigma_i(s_i, d_i)$ in time slot t:

Require: $T_t, \sigma_i(s_i, d_i), \delta \in (0, 2]$

1: $T_{t+1} \leftarrow T_t$;
2: **if** $x_t = LCA_t(s_i, d_i)$ **then**
3: $\quad sendUpdateWeights(T_t, bottom\text{-}up, \sigma_i)$;
4: $splayType \leftarrow getSplayType(T_t, \sigma_i)$;
5: $direction \leftarrow getDirection(T_t, \sigma_i)$;
6: $\Delta\Phi_t \leftarrow deltaPhi(T_t, splayType, direction)$;
7: **if** $\Delta\Phi_t < -\delta$ **then**
8: $\quad T_{t+1} \leftarrow splay(T_t, \sigma_i, direction, splayType)$;
9: **else**
10: $\quad forward(T_t, \sigma_i, direction)$;
11: $updateWeights(T_t, splayType, direction)$;

Algorithm 2 Adaptive CBNet $decayWeight()$ function:

Require: current round t, $timeStamp$, $timeOut$

1: **if** $timeOut < t - timeStamp$ **then**
2: $\quad timeDiff \leftarrow (t - timeStamp)/timeOut$;
3: $\quad W_t \leftarrow W_t >> timeDiff$;
4: $\quad timeStamp \leftarrow t$;

Lemma 1 (Access Lemma (bottom-up [36] and top-down [13] semi-splays))**.** *Consider a $step_t(\sigma_i, x_t)$, executed by the current node $x_t \in T_t$ of rank $r_t(x_t), b_i \leq t \leq e_i$, of type rotation with $splayType \in \{$semi zig-zig, semi zig-zag$\}$. Let $\Delta\Phi_t$ be the net decrease in the potential of T_t, x_{t+1} be the new current node of the message, and $r_{t+1}(x_{t+1})$ be the rank of the latter, right after $step_t$, respectively. If the direction of the step is bottom-up, we have that $x_{t+1} = x_t.p.p$ and $\Delta\Phi_t + 2 \leq 2(r_{t+1}(x_{t+1}) - r_t(x_t))$. Otherwise, if the direction is top-down, we have that $x_t = x_{t+1}.p.p$ and $\Delta\Phi_t + 2 \leq 2(r_t(x_t) - r_{t+1}(x_{t+1}))$.*

The first amortized analysis of self-adjusting networks was presented for

SplayNet [7]. In Theorem 1 we prove that the amortized routing cost of CBNet is asymptotically the same as the amortized reconfiguration cost of SplayNet.

Theorem 1. *Consider any initial BST T_0, a sequence of m messages $\sigma_i \in \sigma$ and Algorithm 1. Let $H(\hat{S})$ and $H(\hat{D})$ be the source and destination empirical entropies of σ, as defined in (4). The* ***amortized routing cost*** *of Algorithm 1 to deliver all messages in σ is $\mathcal{L}_A(CBNet, T_0, \sigma) = O(H(\hat{S}) + H(\hat{D}))$.*

Proof. To bound the routing cost of CBNet, we analyze the number of routing steps performed to deliver a message $\sigma_i(s_i, d_i) \in \sigma$ and the respective weight update message sent from the $LCA_t(s_i, d_i)$ to the root of the tree (recall from Section 5 that this update message carries no data but might trigger rotations). Consider a $step_t(\sigma_i(s_i, d_i), x_t)$ that *has not caused a rotation* in the bottom-up direction (the top-down direction case is analogous), executed by the current node x_t. Since no rotation was performed during this step, by Algorithm 1, $\Delta\Phi_t \geq -\delta, \delta \in (0, 2]$, $\Delta\Phi_t$ being the decrease in total potential of the tree immediately after the referred step. By Lemma 1, we have that: $2 - \delta < 2 + \Delta\Phi_t \leq 2(r_{t+1}(x_{t+1}) - r_t(x_t))$, where $x_{t+1} = x_t.p.p$ and $r_{t+1}(x_{t+1})$ is the rank of the new current node after the step. Let $\delta' = 1 - \delta/2$, $t = e_i$, $\alpha = LCA_t(s_i, d_i)$ and $\rho \in V$ be the root of T_t. Then $r_t(z) - r_t(x) > \delta'$ for each pair of consecutive edges $(x, y) \in T_t$ and $(y, z) \in T_t$ on the (bottom-up) path $\mathcal{P}_t(s, \rho)$ (and analogously on the (top-down) path $\mathcal{P}_t(\alpha, d_i)$). Let $l_i = \lfloor |\mathcal{P}_t(s_i, d_i) \cup \mathcal{P}_t(\alpha, \rho)|/2 \rfloor$. Summing over all l_i pairs of edges, we have that the sum telescopes to $(r_t(\rho) - r_t(s_i)) + (r_t(\alpha) - r_t(d_i)) > l_i\delta'$, and:

$$\begin{aligned} l_i &< \frac{(r_t(\rho) - r_t(s_i)) + (r_t(\alpha) - r_t(d_i))}{\delta'} \leq \frac{(r_t(\rho) - r_t(s_i)) + (r_t(\rho) - r_t(d_i))}{\delta'} \\ &= O\left(\log \frac{W_t(\rho)}{W_t(s_i)} + \log \frac{W_t(\rho)}{W_t(d_i)}\right) = O\left(\log \frac{W(\rho)}{c(s_i)} + \log \frac{W(\rho)}{c(d_i)}\right), \forall \sigma_i \in \sigma \end{aligned}$$

where $W(v)$ and $c(v)$ are the total weight and count of node $v \in V$, respectively. The last equality follows from the fact that $W_t(v) \geq c_t(v), \forall v \in T_t, 0 \leq t \leq e_m$ (Def. (1)).

Using the fact that $W(\rho) = 2m$, since each of the m messages in σ increases two counters in the tree, the source and the destination node's, by one, we have

that the amortized routing cost is

$$\begin{aligned}
\mathcal{L}_A(CBNet, T_0, \sigma) &= \frac{1}{m}\sum_{i=1}^{m} l_i \\
&= O\left(\frac{1}{m}\left(\sum_{i=1}^{m}\log\frac{m}{c(src(\sigma_i))} + \sum_{i=1}^{m}\log\frac{m}{c(dst(\sigma_i))}\right)\right) \\
&= O\left(\frac{1}{m}\left(\sum_{i=1}^{n} s(v_i)\log\frac{m}{c(v_i)} + \sum_{j=1}^{n} d(v_j)\log\frac{m}{c(v_j)}\right)\right) \\
&= O\left(\frac{1}{m}\left(\sum_{i=1}^{n} s(v_i)\log\frac{m}{s(v_i)} + \sum_{j=1}^{n} d(v_j)\log\frac{m}{d(v_j)}\right)\right) \\
&= O(H(\hat{S}) + H(\hat{D})),
\end{aligned}$$

where $s(v_i)$ and $d(v_i)$ are the number of times a node $v_i \in V$ was source and destination in σ, respectively, and $src(\sigma_i(s_i, d_i)) = s_i$ and $dst(\sigma_i(s_i, d_i)) = d_i, \forall \sigma_i \in \sigma$. Note that $c(v_i) \geq s(v_i)$ and $c(v_i) \geq d(v_i), \forall i \in \{1 \dots n\}$. □

Having upper bounded the amortized routing cost, we turn our attention to the reconfiguration cost of CBNet. In Theorem 2 we present a generalization of the analysis of CBTrees [13], a self-adjusting data structure, for self-adjusting networks, while keeping the bound on the number of rotations.

Theorem 2. *Consider any initial BST T_0, a sequence of m messages $\sigma_i \in \sigma$ and Algorithm 1. The* ***total reconfiguration cost*** *incurred by Algorithm 1 to deliver all messages in σ is $\Omega(CBNet, T_0, \sigma) = O\left(n \log \frac{m}{n}\right)$.*

Proof. Each rotation performed by Algorithm 1 decreases the total potential of the tree by at least $\delta \in (0, 2]$. The potential decreases only by means of rotations and cannot be negative, by definition. Hence, the number of rotations, performed by the sequence of all current nodes of each message, is upper bounded by the sum of potential increases throughout the message sequence σ, i.e., the total reconfiguration cost (Def. (2)) $\Omega(CBNet, T_0, \sigma) = O(\Delta\Phi^{+}(\sigma))$. The potential of the network increases when the counters $c_t(s_i)$ and $c_t(d_i)$, $1 \leq t \leq e_m$ are incremented upon the delivery of each message $\sigma_i(s_i, d_i) \in \sigma, 1 \leq i \leq m$, as well as the weights of the nodes that belong to the path traversed by σ_i

and the respective weight-update message sent toward the root. Let $t = e_i$, $\alpha = LCA_t(s_i, d_i)$, $\rho \in V$ be the root of the tree instance T_t, and $\Delta\Phi^+(\sigma_i)$ be the network *potential increase* caused by each message $\sigma_i \in \sigma$, then

$$\begin{aligned}\Delta\Phi^+(\sigma_i) &\leq \sum_{u \in \mathcal{P}_t(s_i,\rho) \cup \mathcal{P}_t(d_i,\alpha)} \log\left(W_t(u)+2\right) - \log W_t(u) \\ &= \sum_{u \in \mathcal{P}_t(s_i,\rho) \cup \mathcal{P}_t(d_i,\alpha)} \log\left(1 + \frac{2}{W_t(u)}\right) \\ &\leq \sum_{u \in \mathcal{P}_t(s_i,\rho) \cup \mathcal{P}_t(d_i,\alpha)} \frac{2}{W_t(u)}, \end{aligned} \tag{3}$$

We can split the sum in (3) into two sums, from s_i to ρ and from d_i to α:

$$\Delta\Phi^+(\sigma_i) \leq \sum_{u \in \mathcal{P}_t(s_i,\rho)} \frac{2}{W_t(u)} + \sum_{v \in \mathcal{P}_t(d_i,\alpha)} \frac{2}{W_t(v)}. \tag{4}$$

As we argued in Theorem 1, for every pair of consecutive edges $(x, y), (y, z)$ on the path traversed by message σ_i, $W_t(z)/W_t(x) > 2^{\delta'}, \delta' \in (0, 1]$. It follows that $2/W_t(u)$ for $u \in \mathcal{P}_t(s_i, \rho)$ and $\mathcal{P}_t(d_i, \alpha)$ are both geometric series. So the two summation terms in (4) converge to $O(1/W_t(s_i))$ and $O(1/W_t(d_i))$, respectively.

Now consider all communication requests $\sigma_i \in \sigma$. Let $v \in V$ be a particular node in the network. After the k-th time v has been a source or a destination of some message in σ, its counter becomes $c_t(v) = k$. Thus, the potential increase due to all communications over $v \in V$ is

$$\begin{aligned}\Delta\Phi^+(\sigma) &= \sum_{i=1}^{m} O\left(\frac{1}{c_{e_i}(src(\sigma_i))} + \frac{1}{c_{e_i}(dst(\sigma_i))}\right) \\ &= \sum_{v \in V} \sum_{j | v = src(\sigma_j) || v = dst(\sigma_j)} O\left(\frac{1}{c_{e_j}(v)}\right) \\ &= \sum_{v \in V} O(\log c(v)) = O\left(n \log \frac{m}{n}\right) \end{aligned} \tag{5}$$

where (5) follows from the fact that $\sum_{v \in V} \log c(v)$ is maximized when $c(v) = m/n, \forall v \in V$.

□

Theorem 3. *Consider any initial BST T_0, a sequence of m messages $\sigma_i \in \sigma$ and Algorithm 1. The* ***total work cost*** *incurred by Algorithm 1 to deliver all messages in σ is $\mathcal{C}(CBNet, T_0, \sigma) = O\left(m \log n + n \log \frac{m}{n}\right)$.*

Proof. The result follows from the Definition 2 of $\mathcal{C}(\mathcal{A}, T_0, \sigma)$, the bounds $\mathcal{L}(CBNet, T_0, \sigma) = O(H(\hat{S}) + H(\hat{D}))$ (Theorem 1) and $\Omega(CBNet, T_0, \sigma) = O\left(n \log \frac{m}{n}\right)$ (Theorem 2), and the fact that the empirical entropy (Definition 4) is maximized when the source and destination distributions are uniform, i.e., it is $O(\log n)$. □

Since the execution of Algorithm 1 is sequential, its time cost has the same asymptotic upper bound as its total communication cost.

8. Concurrent CBNet

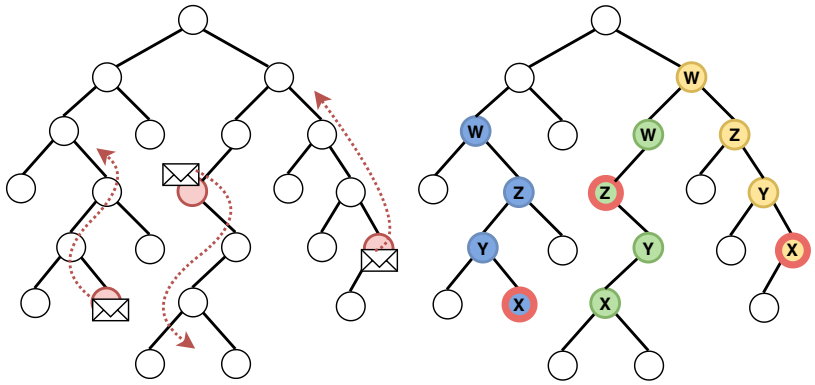
We now turn our attention to the concurrent CBNet implementation and analysis. Firstly, we highlight some of the assumptions made when modeling concurrent network communication. In Section 8.1 we explain the main algorithm design principles that we used. In Section 8.2, we prove the liveness of and analyze the communication cost incurred by the concurrent CBNet.

8.1. Concurrent network reconfiguration

To implement the concurrent CBNet we adopt the design principles of DiSplayNet [12], which follow an optimistic approach, i.e., conflicts are dealt with upon occurrence, and can be summarized as follows:
(1) **Prioritization:** In order to ensure deadlock and starvation freedom, concurrent steps are executed according to a *priority*. Given two messages σ_i and σ_j, we say that σ_i has a higher priority than σ_j if $b_i < b_j$;
(2) **Independent clustering:** To ensure consistency among concurrent steps, local independent clusters of nodes that participate in a single step are computed in a distributed manner. A cluster is formally defined below.

Definition 6. ***Cluster*** $\mathcal{K}_t(\sigma_i, x_t)$***:*** *Consider the set of links* $(v, w) \in T_t$ *that would be modified as a result of a rotation executed by a* $step_t(\sigma_i, x_t)$*. The* ***parent node*** v *of each such link belongs to a set of nodes that is referred to as the cluster* $\mathcal{K}_t(\sigma_i, x_t)$*. In each cluster, the current node of the respective step,* $x_t \in T_t$*, is the only node that can perform a step, even if it is decided that it will be of type routing; the other nodes in the cluster are* locked *in round* t*.*

Figure 1 illustrates the concurrent execution of 3 non conflicting clusters.



(a) Bottom-up and top-down steps

(b) Splay type determines clusters

(c) Rotations are performed

(d) Messages made progress

Figure 1: Three concurrent not conflicting clusters: (a)-(b) blue and yellow nodes participate in bottom-up steps with current nodes x, green nodes participate in a top-down step with current node z. (c)-(d) Three rotations occurred in parallel, and the three messages have made progress in the tree while staying at the same current nodes.

8.2. Analysis of concurrent execution

In the concurrent scenario, the analysis of CBNet is more challenging than in the sequential setting, because there is only guarantee that the message with the highest priority has consecutive progress towards the destination. For the remaining messages, the consecutive progress can be interrupted, resulting in several consecutive progress sequences. An interruption can cause the message to travel through a different path.

The proof of liveness of CBNet follows the same lines as that of DiSplayNet [11], due to the prioritization rule and the independent clustering approach.

We now turn our attention to the analysis of the communication cost of the concurrent version of CBNet. Firstly, we introduce the concept of a conflict between a pair of concurrent steps.

Definition 7. ***Conflict****: Consider a sequence of messages σ and two messages $\sigma_i = \{s_i, d_i\} \in \sigma$ and $\sigma_j = \{s_j, d_j\} \in \sigma$, such that σ_i has higher priority, i.e., $i < j$. Moreover, consider two steps $step_{t_i}(\sigma_i, x_{t_i}), b_i \leq t_i \leq e_i$ and $step_{t_j}(\sigma_j, x_{t_j}), b_j \leq t_j \leq e_j$ with the respective clusters $\mathcal{K}_{t_i}(\sigma_i, x_{t_i})$ and $\mathcal{K}_{t_j}(\sigma_j, x_{t_j})$. We say that a conflict occurs in the concurrent execution of σ if $t_i = t_j$ and $\mathcal{K}_{t_i}(\sigma_i, x_{t_i}) \cap \mathcal{K}_{t_j}(\sigma_j, x_{t_j}) \neq \emptyset$. A conflict can be of type **pause**, if both steps are of type **routing**, or of type **bypass** if the higher-priority $step_{t_i}(\sigma_i, x_{t_i})$ is of type **rotation**.*

Note that only a bypass can generate a work cost overhead in the network, given that it reconfigures the network topology in the local neighborhood of the current node carrying a message, whereas a pause can generate only a time cost overhead.

Theorem 4. *Consider any initial BST T_0, a sequence of m messages $\sigma_i \in \sigma$ and the concurrent CBNet algorithm (CCBNet). The* ***total reconfiguration cost*** *incurred by CCBNet to deliver all messages in σ is*

$$\Omega(CCBNet, T_0, \sigma) = O\left(n \log \frac{m}{n}\right).$$

Proof. As has been shown in Theorem 2, the number of rotations performed by CBNet is upper bounded by the maximum network potential increase during the execution of σ, which is determined by the total number of messages in σ and is independent of the order or time in which each request $\sigma_i \in \sigma$ is generated or delivered. Given that concurrent CBNet applies the same rule to perform a rotation as the sequential CBNet, the total reconfiguration cost of concurrent CBNet has the same upper bound as in the sequential execution. □

Theorem 5. *Consider any initial BST T_0, a sequence of m messages $\sigma_i \in \sigma$ and the concurrent CBNet algorithm (CCBNet). The* ***total routing cost*** *incurred by CCBNet to deliver all messages in σ is*

$$\mathcal{L}(CCBNet, T_0, \sigma) = O\left(m \log n + n^2 \log \frac{m}{n}\right).$$

Proof. Since only a bypass can generate additional work to deliver a message in σ, and a bypass is associated with at least one rotation, the routing cost overhead due to concurrency is bounded by the total number of rotations in the concurrent execution of σ, multiplied by the additional routing cost incurred by the lower-priority messages that have been bypassed, which is $O(n)$ per bypassed message, in the worst case. The result follows by adding the concurrency overhead to the total sequential routing cost (see Theorem 1 and Def. (3)).

□

Theorem 6. *Consider any initial BST T_0, a sequence of m messages $\sigma_i \in \sigma$ and the concurrent CBNet algorithm (CCBNet). The* ***total work cost*** *incurred by CCBNet to deliver all messages in σ is*

$$\mathcal{C}(CCBNet, T_0, \sigma) = O\left(m \log n + n^2 \log \frac{m}{n}\right).$$

Proof. The result follows from Theorems 1, 4 and 5. □

In Theorem 7 we provide a worst-case upper bound on the time cost of concurrent CBNet.

Theorem 7. *Consider any initial BST T_0, a sequence of m messages $\sigma_i \in \sigma$ and the concurrent CBNet algorithm (CCBNet). The* ***makespan*** *of concurrent CBNet to deliver all messages in σ is*

$$Makespan(CCBNet, T_0, \sigma) = O\left(m \log n + n^2 \log \frac{m}{n}\right).$$

Proof. We know that at least one message (of the highest priority) is delivered without being paused or bypassed at a time in the concurrent execution of CBNet. This implies that the makespan is upper bounded by the total communication cost of the concurrent execution, proved in Theorem 6. □

Comparison to related work. The bounds on work and time costs of concurrent CBNet are a significant improvement compared to the cost of DiSplayNet [11], which has worst-case total work and time costs quadratic in the number of messages:

$$\mathcal{C}(DiSplayNet, T_0, \sigma) = O(m(m+n)\log n).$$

9. Workload traces

In this work, before running each experiment, we measure and classify the locality of the input in terms of its temporal and non-temporal components.

Measuring locality. To measure the locality of reference present in a workload, we use the definition of *trace complexity*, introduced in [1], which leverages only randomization and data compression operations. The amount of locality present in a workload can be measured based on the entropy of the communication sequence. The concept of entropy is related to the amount of information or the ability to compress the data. Intuitively, workloads with a

low locality structure tend to have random sequences of communication pairs and, consequently, a low data compression rate. High-locality sequences tend to have a specific structure between requests that allows for better compression. It is possible to identify and measure the locality present in the dataset leveraging only randomization and data compression operations.

Given a sequence of communication requests σ, we generate two transformations: $\Gamma(\sigma)$, comprised of a sequence of requests where the relative order of requests is shuffled, and temporal relationships are lost; and $\mathcal{U}(\sigma)$, a sequence of requests of the same size and domain, but collected from a uniform distribution, removing the non-temporal locality as a result. The temporal complexity of a trace is given by the ratio between the entropy $C(\sigma)$ contained in the sequence σ and the entropy contained in the temporal transformation $C(\Gamma(\sigma))$: $T(\sigma) = \frac{C(\sigma)}{C(\Gamma(\sigma))}$. Similarly, the non-temporal component of a trace is given by the ratio of the entropy of the temporal transformation $\Gamma(\sigma)$ to the entropy of the uniform sequence $\mathcal{U}(\sigma)$: $NT(\sigma) = \frac{C(\Gamma(\sigma))}{C(\mathcal{U}(\sigma))}$. The entropy $\mathcal{C}(\cdot)$ can be measured through the size of the compressed file. Note that $T, NT \in [0, 1]$, since $C(\sigma) \leq C(\Gamma(\sigma)) \leq C(\mathcal{U}(\sigma)))$.

Definition 8. ***Trace complexity [1]:*** *The complexity of a workload σ is given by the product of temporal and non-temporal complexity: $\Psi(\sigma) = T(\sigma) \times NT(\sigma)$. Note that $\Psi(\sigma) = \frac{C(\sigma)}{C(\Gamma(\sigma))} \times \frac{C(\Gamma(\sigma))}{C(\mathcal{U}(\sigma))}$, and therefore $\Psi(\sigma) = \frac{C(\sigma)}{C(\mathcal{U}(\sigma))}$.*

In Table 1 we list the attributes of all workloads used in our experiments. In Figure 2 we compare these workloads using a graphical representation on a two-dimensional plane. The x and y-axis represent the temporal and non-temporal components, respectively. At each point, the area of the circle corresponds to the trace complexity value $\Psi(\cdot)$. Note that the lower the temporal or non-temporal complexity of a trace, the higher its temporal and non-temporal locality.

In the following, we group the workloads (listed in Table 1) according to their trace complexity and discuss their main characteristics.

High non-temporal and low temporal locality (ProjecToR and Skewed): The ProjecToR workload [46] describes the distribution of communi-

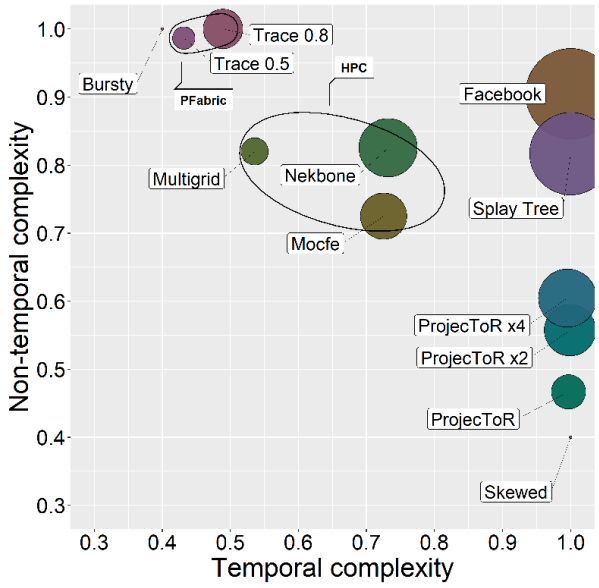


Figure 2: Trace map of all workloads used in the experiments. The x and y axes represent the temporal and non-temporal components, respectively, and the area of each circle corresponds to the trace's complexity, defined in (8).

cation probability between $8,367$ pairs of nodes in a network of 128 nodes (top of racks), running map-reduce and systems storage operations. We sampled a sequence of $10,000$ independent and identically distributed requests (*i.i.d.*) in time and repeated each experiment 30 times. The Skewed workload corresponds to an artificial sequence of $10,000$ communication requests in a network of 1024 nodes, using the method from [1]. The non-temporal locality component was produced using the *Zipf* distribution.

High temporal and low non-temporal locality (PFabric, Bursty): The traces of PFabric [44] were generated by executing simulation scripts in NS2. We sample a sequence of $m = 1,000,000$ requests from a network of 144 nodes. The Bursty workload was generated artificially with $m = 10,000$ and $n = 1024$, using the method from [1].

High temporal and non-temporal locality (HPC [45]): This workload consists of high-performance computing applications, such as solutions to

Table 1: Workload dataset parameters

Scenario	Parameters	Values
	(n, m)	(128, 10,000)
ProjecToR[2] ***i.i.d.***	$\Psi(\sigma) = (T(\sigma), NT(\sigma))$	(0.997, 0.467)
	(n,m)	(1024, 10,000)
Skewed	$(T(\sigma), NT(\sigma))$	(1, 0.4)
PFabric[44] :	(n,m)	(144, 1,000,000)
Trace 0.5	$(T(\sigma), NT(\sigma))$	(0.431, 0.986)
Trace 0.8		(0.489, 1.0)
	(n,m)	(1024, 10,000)
Bursty	$(T(\sigma), NT(\sigma))$	(0.4, 1)
HPC[45] :	(n,m)	(1024, 1,000,000)
Multigrid		(0.535, 0.820)
Nekbone	$(T(\sigma), NT(\sigma))$	(0.731, 0.825)
Mocfe		(0.725, 0.724)
	(n,m)	(159, 1,000,000)
Facebook[3]	$(T(\sigma), NT(\sigma))$	(1.0, 0.904)
	(n,m)	(128, 10,000)
Splay Tree	std	1.6
Data Structure	$(T(\sigma), NT(\sigma))$	(1.0, 0.817)
	(n, m)	(128, 1,000,000)
ProjecToR $\times 2$	$(T(\sigma), NT(\sigma))$	(0.998, 0.557)
ProjecToR $\times 4$		(0.995, 0.604)

Poisson equations. We collected $1,000,000$ requests for a network of 1024 nodes.

Low locality (Facebook and Splay Tree Data Structure): The Facebook [3] trace consists of real *Fbflow*[1] packets collected from the production clusters A of Facebook. The per-packet sampling is uniformly distributed with rate 1:30000, flow samples are aggregated every minute, and node IPs are anonymized. We processed the data as follows. Firstly, we removed all

[1]Network monitoring system that samples packet headers from Facebook's machine fleet.

inter-cluster or intra-rack requests, keeping only inter-rack requests within the same cluster. Then, we globally sorted the requests by timestamp. Finally, we mapped the anonymized IPs to a consecutive value range starting at 0. This resulted in a sequence of $1,000,000$ requests, originated in a 24-hour time window, in a network comprised of 159 nodes. The Splay Tree Data Structure workload consists of an artificial request sequence with $m = 10,000$ and $n = 128$, where each message's destination is the root node of the BST, and the source is chosen randomly, following a normal distribution with $std = 1.6$.

Legacy burden (ProjecToR $\times$ 2, $\times$4): In order to simulate a scenario where a long history of request counting becomes a burden to dynamic self-adjusting, we partitioned the simulation timeline into "epochs" with distinct traffic distributions. We created two workload traces comprised of $1,000,000$ requests. ProjecToR $\times$ 2 is comprised of two parts. The first and the second half of the request sequence were sampled from ProjecToR's distribution. In the second half of the sequence, the ids of the nodes were shuffled. ProjecToR $\times$ 4 was generated similarly, but with four epochs.

10. Simulations

All our experiments were performed using the Sinalgo simulation platform [47], which provides a network abstraction for message passing in a synchronous communication model.

To space the requests in time and make the timestamp sequences more realistic, we used a Poisson distribution with $\lambda = 0.05$ to determine the time of the entrance of each message in the network. We made empirical measurements of the total routing and reconfiguration work and the time required for different communication patterns. In all experiments, we assumed that the reconfiguration cost of one step equals the cost of forwarding a message through one link, i.e., $R = 1$ unit. Note that, in reality, the cost of a reconfiguration is typically much higher than the routing cost, resulting in a significantly higher reduction in total work cost of CBNet than depicted in the simulation plots.

Table 2 lists all the algorithms implemented in our experiments.

Table 2: Simulated algorithms

Baseline	**Description**
CBN	Concurrent CBNet
SCBN	Sequential CBNet
AD1-AD4	Adaptive CBNet ($timeOut \in \{30, 100, 500, 1000\}$ rounds)
BT	A balanced static BST, with no reconfigurations. This baseline corresponds to the optimum topology when all pairs of nodes are equally likely to communicate;
OPT	An optimal static BST, computed using the dynamic program presented in [7]. Like in the BT, there is no reconfiguration cost, but only routing cost. This baseline requires prior knowledge of the request distribution. Note that, since OPT is not a dynamic optimum, it might present a higher routing cost than the dynamic baselines in some scenarios;
SN	Our implementation of SplayNet [7]. Note that SplayNet is not fully distributed, since messages are scheduled sequentially, which would require a global scheduler.
DSN	Our implementation of DiSplayNet, a concurrent version of SplayNet, which is slightly different and more realistic than the one in [11]. We implemented a 3-way handshake procedure for the source and destination nodes of each message to start rotating toward their LCA simultaneously;
SCBN	A sequential version of CBNet. Like SplayNet, it is not entirely distributed and serves as a reference to assess the benefits of concurrent reconfigurations.

10.1. Results

We structure our discussion according to the type of locality of the workloads.

High non-temporal locality (ProjecToR and Skewed): In Figures 3a and 3b we analyze the work cost, which is comprised of the reconfiguration (rotations) and message forwarding (routing) components. In both ProjecToR and Skewed workloads, CBNet performed less (almost no) rotations compared to other self-adjusting networks and the least amount of total work among all baselines except for OPT, due to the low temporal locality of these workloads. The difference in work between BT and OPT is coherent with the presence of non-temporal locality in the traces, as shown in both plots. Compared to BT, it is possible to see that CBNet took advantage of the non-temporal locality present in both workloads, bringing the network closer to the optimal static tree configuration (OPT). CBNet and SCBN had similar total work costs to that of DSN and SN. However, unlike the latter, where most of the work is due to reconfiguration, CBNet performed routing steps almost exclusively, which supports our analytical results.

Figures 4a and 4b present the results in terms of makespan and throughput.[2] Note that we did not include the static networks, as there is no defined time model for them. The first point that we highlight is the higher throughput and lower makespan of the two concurrent versions of self-adjusting networks (CBN and DSN) compared to their sequential counterparts (SCBN and SN), showing that they took advantage of opportunities for parallelism. The second point is the superior performance of CBNet (CBN) compared to DiSplayNet (DSN). This is due to the way CBNet transmits messages. Unlike DSN, where both the source and destination nodes must be dedicated exclusively to a single communication request, CBNet does not lock the destination nodes, so they are able to work on other messages in parallel.

Figures 5a and 5b show the number of concurrent clusters (defined in 6) per round of simulation as a histogram. The main difference between CBNet and DSN is that the former presented significantly less rounds with a single cluster.

[2]Differently from work, throughput is the number of messages delivered per round. Figures 4 and 7 show a smoothed version of the throughput histogram (kernel density estimate).

High temporal locality (PFabric and Bursty): In the PFabric and Bursty workloads, the communication matrix is close to uniform, and consequently, the total work of BT and OPT is almost the same, as can be seen in Figures 3c and 6b. Among all the algorithms, DiSplayNet had the best performance in terms of total work, showing how well this algorithm exploits temporal locality (especially repeated consecutive requests) by performing rotations aggressively. For Bursty, DSN performed even less total work than OPT. In contrast, since CBNet takes into account the entire communication history of the nodes to perform reconfigurations, which makes it less reactive to temporal locality of the workload, CBNet approached the OPT, which in turn approached the BT (balanced BST) work cost.

In Figures 4c and 7b we can see that, in terms of makespan and throughput, however, CBNet performed as well as DiSplayNet, or better, despite the unfavorable trace complexity of the PFabric and Bursty workloads. This shows that CBNet achieved higher parallelism, which can be confirmed by the cluster distribution depicted in Figures 5c and 8b. Note that due to the artificially high temporal locality in the Bursty workload, where the request sequence is comprised of long consecutive request repetitions, the execution becomes sequential at some nodes, limiting opportunities for concurrency.

High non-temporal and temporal locality (HPC): Similarly to the Bursty scenario (3c), DSN and SN performed the least amount of work, even less than OPT, for the HPC Mocfe and Multigrid traces (Figure 6a). Since OPT and BT presented similar values, we can conclude that the non-temporal locality is low, resulting in higher work cost for CBNet and SCBN. Nevertheless, CBNet achieved the best makespan and throughput due to higher parallelism, as shown in Figure 7a and confirmed by Figure 8a, which shows that the number of rounds with ≤ 2 concurrent clusters was almost $10\times$ higher for DSN for the Mocfe and Multigrid traces.

Low locality (Facebook and Splay Tree Data Structure): The results for the Splay Tree Data Structure (Figures 3d, 4d, 5d) and Facebook workloads (Figures 6c, 7c, 8c) were similar for all metrics. Due to the lack of both spatial

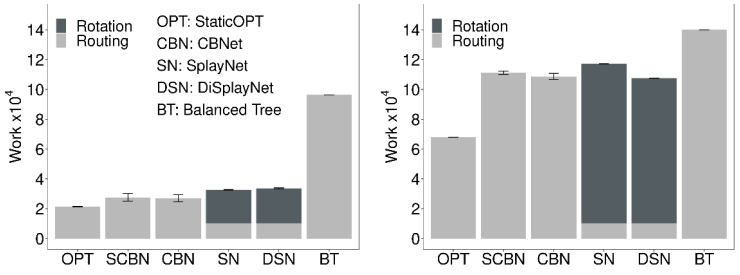
and temporal locality, DSN and SN performed significantly more work among all baselines, more even than the balanced static network (BT), while CBNet performed similarly to OPT, as shown in Figures 3d and 6c. This is due to the lack of spatial locality. For these request sequences, the balanced tree is in fact the optimal topology, so CBNet performed almost no rotations after converging to BST. DSN and SN, however, performed rotations along the entire simulation, due to the lack of temporal locality.

As shown in Figures 4d and 7c, CBNet and SCBN obtained the best results in terms of throughput and makespan. Interestingly, the execution of DSN was sequential for the Slay Tree Data Structure scenario (Figures 4d), which is due to the fact that all requests originate from the root node, preventing any parallelism, as can also be seen in Figure 5d (all DSN rounds had ≤ 1 clusters).

Adaptive CBNet (ProjecToR $\times 2$ and $\times 4$): Finally, we evaluated the performance of Adaptive CBNet in two scenarios, where the demand was comprised of a concatenation of time periods with distinct distributions. We simulated the $decayWeights()$ function (Algorithm 2) with increasing timeout values ($timeOut \in \{30, 100, 500, 1000\}$ rounds). We refer to each configuration as AD1, AD2, AD3, and AD4, respectively.

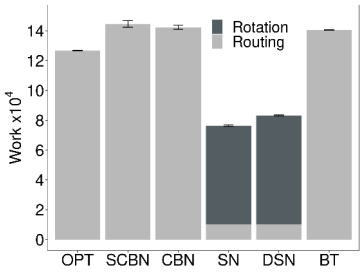
In Figure 9, we observe that CBNet performed more total work than DSN, especially for ProjecToR $\times 4$, due to the high "legacy burden" of this request sequence. Moreover, AD1 performed significantly less work than CBNet and slightly less work than DSN in both scenarios. In ProjecToR $\times 2$, AD2, AD3, and AD4 did not show great advantage compared to CBNet. For ProjecToR $\times 4$, however, all four versions of AD outperformed the non-adaptive CBNet.

This demonstrates that the "legacy burden" of the counting-based self-adjusting approach can be mitigated by a counter resetting scheme, in exchange of slightly higher number of rotations. This increase in reconfiguration cost, however, was insignificant, when compared to the aggressive reconfiguration approach of DSN.

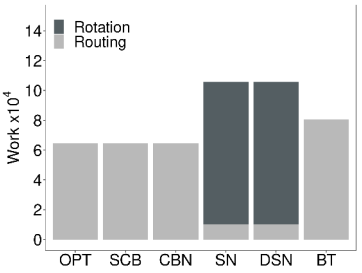


(a) ProjecToR ($\downarrow\downarrow\downarrow$ $T(\sigma)/\uparrow\uparrow$ $NT(\sigma)$)

(b) Skewed ($\downarrow\downarrow\downarrow$ $T(\sigma)/\uparrow\uparrow\uparrow$ $NT(\sigma)$)



(c) Bursty ($\uparrow\uparrow\uparrow$ $T(\sigma)/\downarrow\downarrow\downarrow$ $NT(\sigma)$)



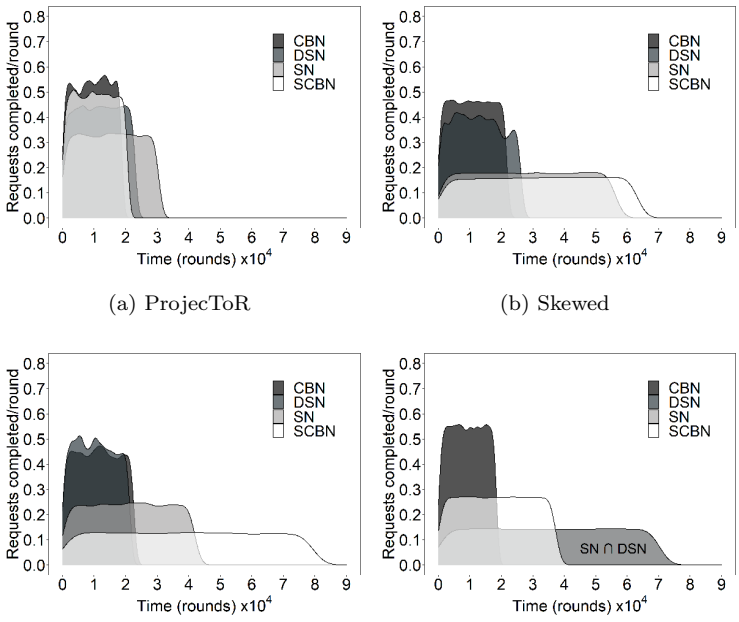
(d) Splay Tree ($\downarrow\downarrow\downarrow$ $T(\sigma)/\uparrow$ $NT(\sigma)$)

Figure 3: Work cost for synthetic workload traces ($m = 10,000$)

11. Conclusion

We presented CBNet, a self-adjusting network that relies on a fully decentralized control plane and significantly reduces adjustment costs, compared to prior work. Despite its concurrent nature, CBNet comes with formal guarantee.

In this work, we used a synchronous distributed system model. CBNet assumes a consistent and communication-closed round structure provided by the distributed system (i.e., if a message is sent by a node, it will be delivered to the receiver node in the same round). While a large number of solutions to problems in distributed computing assume lock-step rounds, real-world distributed systems are usually not perfectly synchronous. In practice, message loss is present as a result of job dropping by a real-time scheduler or an unreliable communication channel. There are several ways to generate round structures in various different systems and failure models, e.g., [48]. One way to overcome this issue is to build a synchronous round abstraction atop the imperfect communication



(a) ProjecToR (b) Skewed

(c) Bursty (d) Splay Tree

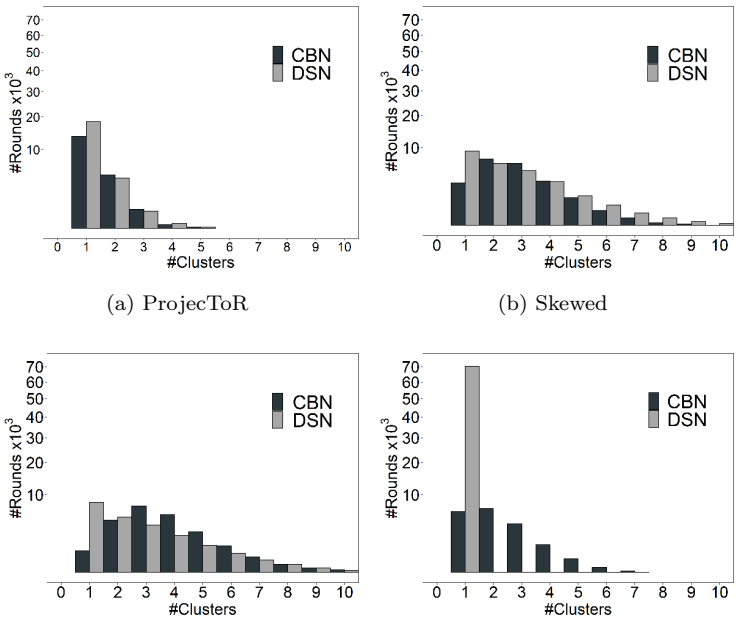
Figure 4: Throughput and makespan for synthetic workload traces ($m = 10,000$)

system, using a synchronizer [49]. The real-time performance of an atop running synchronous algorithm depends on the performance of the synchronizer, as well as on the assumptions w.r.t. delay bounds and fault probability distribution.

While our contribution is still theoretical in nature, we believe that it constitutes an interesting step forward toward practical self-adjusting networks. Our work opens interesting avenues for future research. In particular, it will be interesting to further analyze the optimal trade-off between the benefits and the costs of adjustments. It will also be interesting to generalize the network topology beyond trees and consider implications on network and transport layers.

Acknowledgments

This research work was supported by CAPES, CNPq, Fapemig, and the European Research Council (ERC), grant agreement No. 864228 (AdjustNet), Horizon 2020, 2020-2025.



(a) ProjecToR (b) Skewed

(c) Bursty (d) Splay Tree

Figure 5: Synthetic workloads ($m = 10,000$): number of concurrent clusters per round

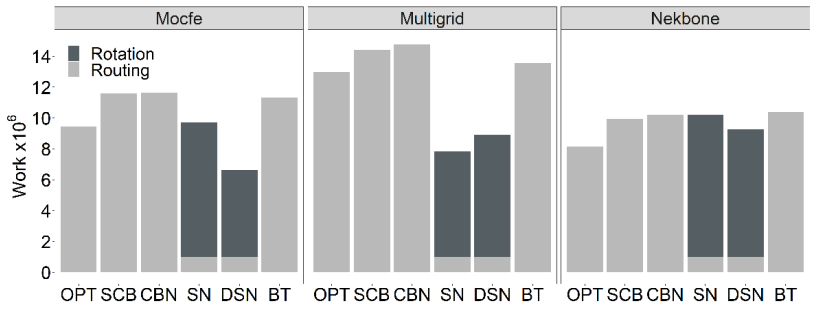
References

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

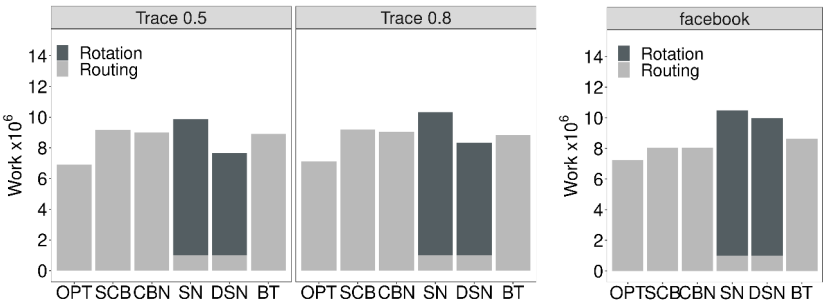
[2] M. Ghobadi, R. Mahajan, A. Phanishayee, N. Devanur, J. Kulkarni, G. Ranade, P.-A. Blanche, H. Rastegarfar, M. Glick, D. Kilper, Projector: Agile reconfigurable data center interconnect, in: Proceedings of the 2016 ACM SIGCOMM Conference, ACM, 2016, pp. 216–229.

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

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



(a) HPC ($\uparrow\uparrow T(\sigma)/\uparrow NT(\sigma)$)



(b) PFabric ($\uparrow\uparrow\uparrow T(\sigma)/\downarrow\downarrow\downarrow NT(\sigma)$)

(c) Facebook $\downarrow\downarrow\downarrow T/\ NT$

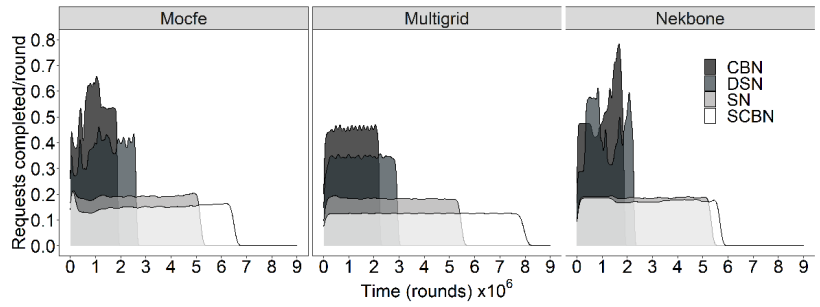
Figure 6: Real-system workload traces ($m = 1,000,000$): total work cost

[5] S. Kandula, J. Padhye, P. Bahl, Flyways to de-congest data center networks, Proc. ACM HotNets (2009).

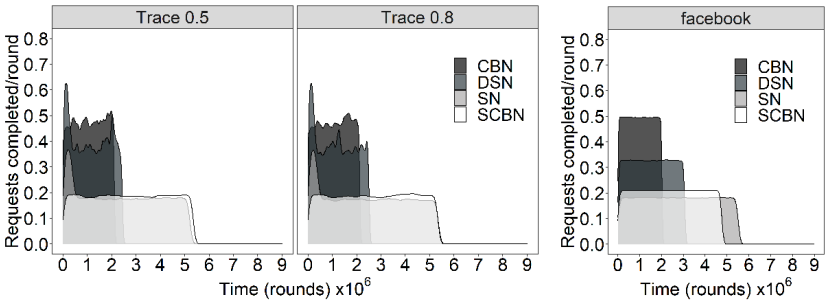
[6] C. Avin, S. Schmid, Toward demand-aware networking: A theory for self-adjusting networks, in: ACM SIGCOMM Computer Communication Review (CCR), 2018.

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

[8] N. Farrington, G. Porter, S. Radhakrishnan, H. H. Bazzaz, V. Subramanya, Y. Fainman, G. Papen, A. Vahdat, Helios: a hybrid electrical/optical switch architecture for modular data centers, ACM SIGCOMM Comp. Comm. Review 41 (4) (2011) 339–350.



(a) HPC



(b) PFabric (c) Facebook

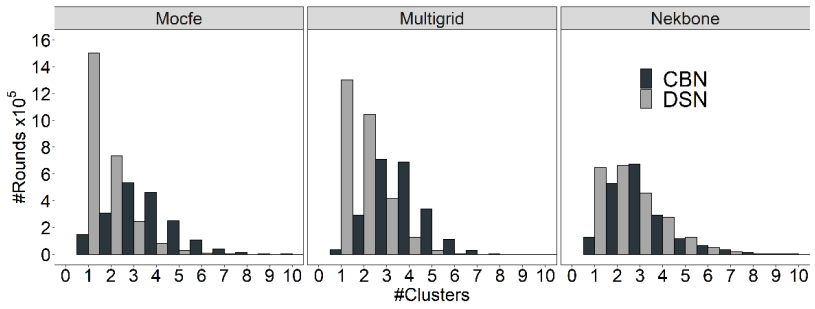
Figure 7: Real-system workload traces ($m = 1,000,000$): throughput and makespan

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

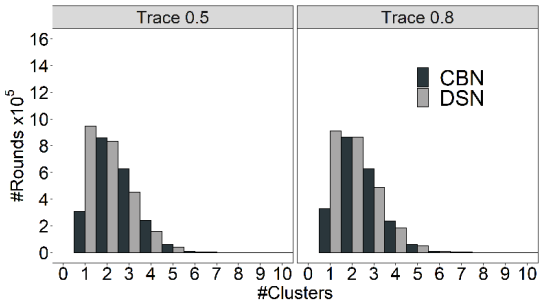
[10] N. Hamedazimi, Z. Qazi, H. Gupta, V. Sekar, S. R. Das, J. P. Longtin, H. Shah, A. Tanwer, Firefly: A reconfigurable wireless data center fabric using free-space optics, in: ACM SIGCOMM Computer Communication Review, Vol. 44, ACM, 2014, pp. 319–330.

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

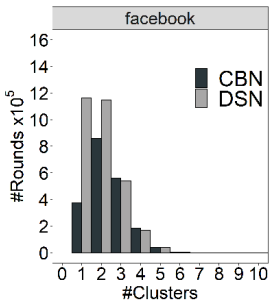
[12] B. Peres, O. A. d. O. Souza, O. Goussevskaya, C. Avin, S. Schmid, Distributed self-adjusting tree networks, IEEE Tran. on Cloud Computing (2021).



(a) HPC



(b) PFabric



(c) Facebook

Figure 8: Real-system workloads ($m = 1,000,000$): Number of concurrent clusters per round

[13] Y. Afek, H. Kaplan, B. Korenfeld, A. Morrison, R. E. Tarjan, The cb tree: A practical concurrent self-adjusting search tree, Distrib. Comput. 27 (6) (2014) 393–417.

[14] M. Al-Fares, A. Loukissas, A. Vahdat, A scalable, commodity data center network architecture, in: ACM SIGCOMM Comp. Comm. Review, Vol. 38, ACM, 2008, pp. 63–74.

[15] 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, ACM SIGCOMM Comp. Comm. Review 45 (4) (2015) 183–197.

[16] V. Liu, D. Halperin, A. Krishnamurthy, T. Anderson, F10: A fault-tolerant

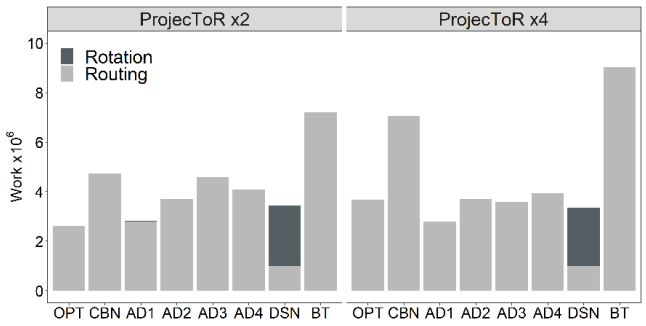


Figure 9: Adaptive CBNet (AD1-AD4: $timeOut \in \{30, 100, 500, 1000\}$ rounds): total work

engineered network, in: USENIX Symposium on Networked Systems Design and Implementation (NSDI), 2013, pp. 399–412.

[17] C. Guo, G. Lu, D. Li, H. Wu, X. Zhang, Y. Shi, C. Tian, Y. Zhang, S. Lu, Bcube: a high performance, server-centric network architecture for modular data centers, Proc. ACM SIGCOMM Computer Communication Review (CCR) 39 (4) (2009) 63–74.

[18] H. Wu, G. Lu, D. Li, C. Guo, Y. Zhang, Mdcube: a high performance network structure for modular data center interconnection, in: Proc. ACM CONEXT, 2009, pp. 25–36.

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

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

[21] W. M. Mellette, R. McGuinness, A. Roy, A. Forencich, G. Papen, A. C. Snoeren, G. Porter, Rotornet: A scalable, low-complexity, optical datacenter network, in: Proc. ACM SIGCOMM, 2017, pp. 267–280.

[22] W. M. Mellette, R. Das, Y. Guo, R. McGuinness, A. C. Snoeren, G. Porter,

Expanding across time to deliver bandwidth efficiency and low latency, arXiv preprint arXiv:1903.12307 (2019).

[23] 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, pp. 782–797.

[24] K. Chen, A. Singla, A. Singh, K. Ramachandran, L. Xu, Y. Zhang, X. Wen, Y. Chen, Osa: An optical switching architecture for data center networks with unprecedented flexibility, IEEE/ACM Transactions on Networking 22 (2) (2014) 498–511.

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

[26] J. Kulkarni, S. Schmid, P. Schmidt, Scheduling opportunistic links in two-tiered reconfigurable datacenters, in: 33rd ACM Symposium on Parallelism in Algorithms and Architectures (SPAA), 2021.

[27] C. Avin, A. Hercules, A. Loukas, S. Schmid, rdan: Toward robust demand-aware network designs, in: Information Processing Letters (IPL), 2018.

[28] S. B. Venkatakrishnan, M. Alizadeh, P. Viswanath, Costly circuits, submodular schedules and approximate carathéodory theorems, Queueing Systems 88 (3-4) (2018) 311–347.

[29] G. Porter, R. Strong, N. Farrington, A. Forencich, P. Chen-Sun, T. Rosing, Y. Fainman, G. Papen, A. Vahdat, Integrating microsecond circuit switching into the data center, SIGCOMM Comput. Commun. Rev. 43 (4) (2013) 447–458.

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

[31] E. Feder, I. Rathod, P. Shyamsukha, R. Sama, V. Aksenov, I. Salem, S. Schmid, Lazy self-adjusting bounded-degree networks for the matching model, in: Proc. IEEE Conference on Computer Communications (INFOCOM), 2022.

[32] C. Griner, J. Zerwas, A. Blenk, S. Schmid, M. Ghobadi, C. Avin, Cerberus: The power of choices in datacenter topology design (a throughput perspective), in: Proc. ACM SIGMETRICS, 2022.

[33] L. Chen, K. Chen, Z. Zhu, M. Yu, G. Porter, C. Qiao, S. Zhong, Enabling wide-spread communications on optical fabric with megaswitch, in: Proceedings of the 14th USENIX Conference on Networked Systems Design and Implementation, NSDI'17, USA, 2017, pp. 577–593.

[34] G. Wang, D. G. Andersen, M. Kaminsky, K. Papagiannaki, T. Ng, M. Kozuch, M. Ryan, c-through: Part-time optics in data centers, ACM SIGCOMM Computer Communication Review 41 (4) (2011) 327–338.

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

[36] D. Sleator, R. Tarjan, Self-adjusting binary search trees, Journal of the ACM (JACM) 32 (3) (1985) 652–686.

[37] J. Zerwas, W. Kellerer, A. Blenk, What you need to know about optical circuit reconfigurations in datacenter networks, in: The 33nd International Teletraffic Congress (ITC 33), 2021.

[38] M. Wang, Y. Cui, S. Xiao, X. Wang, D. Yang, K. Chen, J. Zhu, Neural network meets dcn: Traffic-driven topology adaptation with deep learning, Proc. ACM Meas. Anal. Comput. Syst. 2 (2) (2018).

[39] L. Chen, K. Chen, Z. Zhu, M. Yu, G. Porter, C. Qiao, S. Zhong, Enabling wide-spread communications on optical fabric with megaswitch, NSDI'17, 2017, p. 577–593.

[40] J. Zerwas, C. Avin, S. Schmid, A. Blenk, exrec: Experimental framework for reconfigurable networks based on off-the-shelf hardware.

[41] Polatis Series 6000, www.polatis.com, [Online; accessed 10-May-2022].

[42] B. Allen, I. Munro, Self-organizing binary search trees, J. ACM 25 (4) (1978) 526–535.

[43] O. A. d. O. Souza, O. Goussevskaia, S. Schmid, Cbnet: Minimizing adjustments in concurrent demand-aware tree networks, in: IEEE Int. Parallel and Distributed Processing Sym. (IPDPS), 2021, pp. 382–391.

[44] M. Alizadeh, S. Yang, M. Sharif, S. Katti, N. McKeown, B. Prabhakar, S. Shenker, Pfabric: Minimal near-optimal datacenter transport, in: Proc. ACM Conference on SIGCOMM, 2013, p. 435–446.

[45] U. DOE, Characterization of the doe mini-apps., https://portal.nersc.gov/project/CAL/overview.htm, accessed 11-February-2020 (2016).

[46] Projector dataset, www.microsoft.com/en-us/research/project/projector-agile-reconfigurable-data-center-interconnect (2016).

[47] D. C. Group, Sinalgo - simulator for network algorithms, http://disco.ethz.ch/projects/sinalgo, accessed 10-Jan-2020 (2007).

[48] O. Bakr, I. Keidar, Evaluating the running time of a communication round over the internet, in: PODC, ACM, 2002, pp. 243–252.

[49] B. Awerbuch, Complexity of network synchronization, J. ACM 32 (4) (1985) 804–823.