LEGUP: Using Heterogeneity to Reduce the Cost of Data Center Network Upgrades

Andrew R. Curtis, S. Keshav and Alejandro Lopez-Ortiz
Cheriton School of Computer Science
University of Waterloo
{a2curtis, keshav, alopez-o}@uwaterloo.ca

ABSTRACT

Fundamental limitations of traditional data center network architectures have led to the development of architectures that provide enormous bisection bandwidth for up to hundreds of thousands of servers. Because these architectures rely on homogeneous switches, implementing one in a legacy data center usually requires replacing most existing switches. Such forklift upgrades are typically prohibitively expensive; instead, a data center manager should be able to selectively add switches to boost bisection bandwidth. Doing so adds heterogeneity to the network's switches and heterogeneous high-performance interconnection topologies are not well understood. Therefore, we develop the theory of heterogeneous Clos networks. We show that our construction needs only as much link capacity as the classic Clos network to route the same traffic matrices and this bound is the optimal. Placing additional equipment in a highly constrained data center is challenging in practice, however. We propose LEGUP to design the topology and physical arrangement of such network upgrades or expansions. Compared to current solutions, we show that LEGUP finds network upgrades with more bisection bandwidth for half the cost. And when expanding a data center iteratively, LEGUP's network has 265% more bisection bandwidth than an iteratively upgraded fat-tree.

1. INTRODUCTION

Most current data center networks use 1+1 redundancy in a three-level tree topology, which provides inadequate bisection bandwidth to achieve *agility*—the ability to assign any server to any service. This reduces server utilization when workloads vary rapidly because dynamic reallocation of services to servers is impractical, so a service is assigned enough servers to handle its peak load. Recent work has addressed this problem by providing enormous bisection bandwidth for up to hundreds of thousands of servers [2, 10, 12, 13, 29]. However, these solutions assume homogeneous switches, each with a prescribed number of ports. Therefore, adopting these solutions in a legacy data center often comes at the cost of replacing nearly all switches in the network and rewiring it. This is wasteful and usually infeasible due to sunk capital costs, downtime, and a slow time to market.

The goal of our work is to allow a data center operator to incrementally add equipment to boost bisection bandwidth and reliability without needing to throw out their existing network. However, this results in the creation of heterogeneous data center network topologies, which have not been sufficiently studied in past work. Therefore, we provide the theoretical foundations of heterogeneous Clos networks. Our construction is provably optimal in that it uses the minimal amount of link capacity possible to meet the hose traffic constraints, which accounts for any traffic matrix supported by the top-of-rack switch uplink rates. Previous work has only considered heterogeneous interconnection networks under a different traffic model [27], which is not applicable to data center networks; we discuss the differences in Sec. 8. To our knowledge, our construction is the first topology that achieves optimality for the hose traffic constraints while supporting switches with heterogeneous rates and numbers of ports.

We then construct a system we call LEGUP to design network upgrades and expansions for existing data centers. LEGUP aims to design an upgraded network that is realizable in a highly constrained data center, while maximizing performance by building a heterogeneous Clos network from existing and new switches. Supporting heterogeneous switches allows LEGUP to design upgrades with significantly more bisection bandwidth than existing techniques for the same dollar cost, which includes the costs of new switches and rewiring the network.

Our key contributions are:

- Development of theory to construct optimal heterogeneous Clos topologies (§4).
- The LEGUP system to design legacy data center network upgrades and expansions with maximal performance, defined here as agility, reliability, and flexibility, subject to a budget and constraints of the existing data center (§3). LEGUP reuses existing networking equipment when possible, minimizes rewiring costs, and selects the location of new equipment.
- We evaluate LEGUP by using it to find network upgrades for a 7,600 server data center based on the Uni-

ACM CoNEXT 2010, November 30–December 3 2010, Philadelphia, USA.

versity of Waterloo's School of Computer Science data center (§6). LEGUP finds a network upgrade with nearly three times more bisection bandwidth for the same dollar cost as a fat-tree or naive scale-out upgrades. LEGUP outperforms other upgrade approaches even when spending half as much money. We also find that when adding servers to a data center in an iterative fashion, the network found by LEGUP has 265% more bisection bandwidth than a similarly upgraded fat-tree after the number of servers is doubled.

Before describing LEGUP and our results, we describe background material (§2). We end with a discussion of our work (§7), related work (§8), and conclusions (§9).

2. BACKGROUND

The switching fabric of most existing data center networks (DCNs) is a 1+1 redundant tree with three levels: the top-of-rack (ToR) switches connect to a level of aggregation switches which connect to a core level made up of either switches or routers. The core level is connected to the internet using edge routers. This architecture has two major drawbacks—poor reliability and insufficient bisection bandwidth—besides many other minor problems, as detailed by Greenberg *et al.* [10, 11].

These limitations have been the focus of much recent work and researchers have proposed a variety of topology constructions. Some current DCN proposals are based on classic network topologies such as fat-trees [2], the Clos network [10], and hypercubes [29]. Others employ novel recursive constructions [1, 12, 13]. These proposals, however, have a common feature: they are highly regular and require homogeneous switches, each with a prescribed number of ports. This makes it nearly impossible to implement them as an upgrade to an existing data center without replacing most switches in the network because each configuration scales to a maximum number of end-hosts. Once that limit is reached, most switches must be replaced with a higher radix (number of ports) switch to support additional servers.

High DCN bisection bandwidth is of primary importance due to the unpredictable nature of DCN traffic. The studies of DCN traffic to date demonstrate that DCNs exhibit highly variable traffic [4, 10, 18]. The traffic matrix (TM) in a DCN shifts frequently and its overall volume changes dramatically in short time periods. Over longer time periods, DCN traffic shows a clear diurnal pattern: traffic peaks during the day and falls off at night (see, e.g., [14]).

Given these traffic patterns, an ideal DCN should be able to *feasibly route* all traffic matrices (TMs) that are possible given the uplink rates of the servers. That is, no link should ever have higher utilization than 1, no matter the server-to-server traffic matrix. The set of TMs allowed under this model is known as the *hose traffic matrices* and was introduced in the context of provisioning virtual private networks [7]. We find it more convenient to deal with the ToR-to-ToR traffic matrix, which aggregates the servers connected to a ToR switch into a single entry. We denote the sum of uplink rates on a ToR switch i by $r(i)$ and call this the *rate* of the switch.

3. LEGUP OVERVIEW

LEGUP guides operators when upgrading or expanding their data center. To achieve this goal, LEGUP solves a network design optimization problem that maximizes performance subject to a budget and the data center's physical constraints. We define data center network performance more precisely next (§3.1), and then give details about the inputs, constraints, and outputs of LEGUP (§3.2). We end this section by giving an overview of the optimization engine used by LEGUP (§3.3).

3.1 Optimization goals

LEGUP designs a network upgrade that maximizes *performance*, which we define it to be a weighted, linear combination of the following metrics:

Agility Rather than focusing on bisection bandwidth, as previous work has done, we focus on the more general concept of agility, which we define to be the maximal constant p_a such that the network can feasibly route all hose traffic matrices (denoted by $\mathcal{D}$) in $p_a \cdot \mathcal{D}$, where each hose TM $D \in \mathcal{D}$ is multiplied by the scalar p_a. Here, p_a can be interpreted as the fraction of servers that can send/receive at their maximum rate regardless of the destination/source and regardless of the TM. A network with no oversubscribed links has an agility of 1. As an example, consider a network consisting of a two switches, each attached to 48 servers at 1Gbps and a single 10Gbps port that connects the switches. The agility of this network is $10/48$. More generally, if we have n servers attached to the first switch and m attached to the second, then we have the agility of the network is $\min\{1, 10/\min\{n, m\}\}$. Here, we divide by the minimum of the two values because the hose TMs do not allow any server to send or *receive* more than 1Gbps of traffic, that is, even if there are 48 servers attached to one switch and 1 server attached to the other, the maximum receiving rate of lone server is 1Gbps so no more than that will ever cross the connecting 10Gbps link.

Flexibility We say that a δ attachment point, is an unused port such that that attaching a 1 unit (in this paper, this is 1Gbps) uplink device to this port does not decrease the network's agility to less than δ. Then, a network is (p_f, δ)-flexible if it has p_f distinct δ attachment points when the attachment points are filled according to some rule (e.g., by greedily assigning devices to the attachment point that lowers agility the minimal amount). As an example, again consider our two switch network, except now assume all 48 of each switch's 1Gbps ports are free. If we take $\delta = 0.5$, then the flexibility of this network is 68, achieved by attaching 48 servers to one switch and 20 to the other. If we attach an additional server to the second switch, then the agility drops to $10/\min\{21, 48\}$ which is less than 0.5.

Reliability Reliability is the number of link or switch failures needed to partition the ToR switches, which we denote by p_r. This model corresponds to the failure of a switch or port or a cable cut. As an example, the complete graph on n vertices has a reliability of $n-1$ because every edge neighboring a vertex must be removed in order to partition the complete graph. The worst case reliability is that of a tree: removing a single node or edge partitions it.

These metrics measure distinct aspects of a network. Agility and reliability are related—increased reliability can increase agility—however, two networks can have the same agility with completely different reliability metrics since link speeds can vary by orders of magnitude. Similarly, high agility is a prerequisite to high flexibility, but switches also must have unused ports for a network to be flexible. We have defined these metrics so that they are computable in polynomial time; we will describe how to compute each later when describing LEGUP's details in Section 5.

3.2 Inputs, Constraints, and Outputs

As **input**, LEGUP requires a budget, a list of available switches and line cards, and a data center model. The budget is the maximum amount of a money that can be spent in the upgrade, and therefore acts as a constraint in the optimization procedure. The available switches are the details and prices of switches that can be purchased. Relevant details for a switch include its ports and their speeds, line card slots (if a modular switch), power consumption, rack units, and thermal output. Details of a line card are its ports, price, and a list of interoperable switches.

Providing a model of the existing data center is optional, and even when provided, can include varying levels of detail. A complete model includes the full details of the network plus the physical arrangement of racks, the contents of each rack, and the power and thermal characteristics of equipment in the racks. Additionally, thermal and power constraints can be included in this description, e.g., the equipment in each rack cannot draw more than 10 kW of power. Details of the existing network includes information about its switches and their locations. Therefore, the per rack physical constrains that LEGUP models are thermal, power, and free rack slots. If details of the existing switches are provided, they will be considered for use in the upgraded network. LEGUP will find a solution, if one exists, that meets the physical constraints given and will minimize the number and length of cable runs.

As **output**, LEGUP gives a detailed blueprint of the upgraded network. This includes its topology and a selection of switches and line cards to obtain. If a data center model was included in the input, LEGUP also outputs the rack where each aggregation switch should be placed and a wiring diagram that specifies each cable run between ToR and aggregation switches.

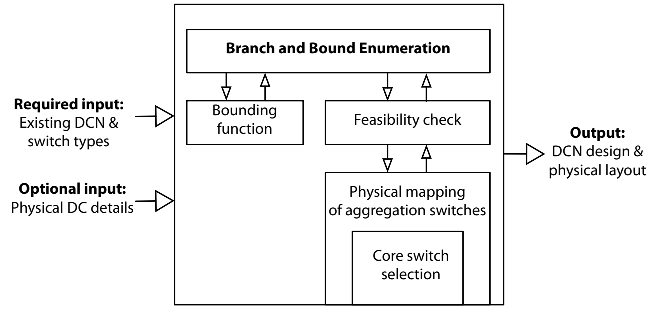


Figure 1: Overview of the LEGUP optimization engine.

3.3 The LEGUP optimization engine

We now give a high level overview of the engine employed by LEGUP. The optimization problem solved by LEGUP maximizes the sum of agility, reliability, and flexibility, weighting each metric by a multiplier selected by the user. This is a difficult optimization problem and is made harder by the large number of constraints (e.g., energy, thermal, space, and cabling).

LEGUP only designs tree-like networks, which is desirable in a data center because many DCN load balancing, routing, and addressing solutions require a tree-like network, e.g., [2, 10, 25]. However, the theory of heterogeneous tree-like topologies has not been previously developed, and we wish to use heterogeneity to reduce the cost of network upgrades. Therefore, we develop the theory of heterogeneous Clos networks in the next section, which are tree-like networks. The reasoning behind this decision is that that a traditional 1+1 redundant DCN topology is already a Clos network instance (albeit a 1+1 redundant topology is a Clos instance that does not have the agility and reliability typically associated with Clos networks). Despite adding heterogeneous switches, DCN addressing and routing solutions can be used on our constructions with no or minor modifications; we discuss this further in Sec. 7.

We assume that all servers already connect to a sufficient ToR switch, but that the aggregation and core levels of the network need to be upgraded. Given a set of aggregation switches, the optimal set of core switches is somewhat restricted in a heterogeneous Clos network, so LEGUP explores the space of aggregation switches using a branch and bound optimization algorithm.

Branch and bound is a general optimization algorithm that finds an optimal solution by enumerating the problem space; however, it achieves efficiency by bounding, and therefore not enumerating, large portions of the problems space that cannot contain an optimal solution. Our branch and bound differs slightly from the standard implementation because we enumerate over only the aggregation switches, so we must introduce additional steps to find a set of core switches; Figure 1 depicts our design.

In our context, the problem space is all possible sets of aggregation switches given the available switch types given as input. We need to build a tree of *candidate solutions*, i.e., the set of aggregation switches used in the network. We call this tree the *solution tree*. Each node in the solution tree is labeled by the set of aggregation switches it represents; the root's label is empty. A node is branched by giving it a child for each switch type; the label of the child is the label of its parent plus the switch type the child represents. A solution is a *complete solution* when its aggregation switches have enough ports to connect the ToR switches with a spanning tree.

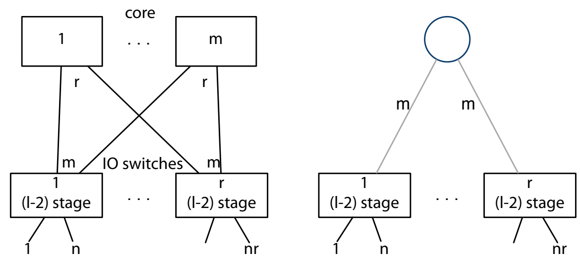
A complete solution only describes the set of aggregation switches in the network and does not account for the core switches nor the physical layout of the network. Given a complete solution, we find the min-cost mapping of solution's aggregation switches to racks (full details of LEGUP's handling of complete solutions are given later in §5) and then find the min-cost set of core switches to connect the aggregation switches to. Once this is complete, we add the cost of the core and physical mapping into the cost of the solution to determine if it is still feasible, i.e., it is not over budget; additionally, we check to make sure no physical constraints (e.g., thermal and power draw) are violated in the physical mapping phase. Unlike standard branch and bound, we continue to branch complete solutions because a solution is complete here whenever it can connect all the ToR switches; however, adding more aggregation switches to a complete solution will always improve its performance (but may violate some constraints).

Before checking for feasibility; however, a candidate solution is bounded to check if it, or any of its children, can be an optimal solution. A candidate is bounded by finding the maximal agility, flexibility, and reliability possible for any solution in its subtree. A candidate solution with a lower bound than the optimal complete solution is trimmed, that is, it is not branched because its subtree cannot possibly contain an optimal solution. We delay the details of our particular bounding function until Section 5.1.

3.4 Why naive solutions aren't enough

To motivate our design of LEGUP, we briefly address the need for algorithms more sophisticated than standard heuristics, e.g., a greedy algorithm. We identify three key weaknesses of existing heuristics that LEGUP addresses:

1. Standard techniques don't take physical constraints into account, and therefore might not return a feasible solution. LEGUP finds a feasible solution if one exists.
2. Algorithms that greedily add switches with the minimum bandwidth to price ratio will always reuse existing switches. This might not be the optimal network configuration. LEGUP only reuses switches when it's beneficial to do so.
3. Cabling and switch costs need to be accounted for. We are unaware of any simple algorithms that take both these costs into account.



(a) Physical realization (b) Logical topology

Figure 2: An l-stage Clos network. Each IO switch here is a subnetwork with $l-2$ stages. In (b), each logical edge represents m physical links and the logical root represents m switches, each with r ports.

Our implementation of LEGUP's branch and bound algorithm uses depth-first search and when it branches a solution tree node, it orders the children so that they are sorted by bandwidth to price ratio. As a result, the first solutions explored by the branch and bound are the solutions that a greedy algorithm considers. We have found this to increase the number of trimmed subtrees dramatically since the first complete solutions tend to have good, though not optimal, performance.

4. THEORY

Our implementation of LEGUP designs heterogeneous Clos networks, so we develop this theory before describing the details of LEGUP's implementation. Before presenting our heterogeneous Clos construction (§4.2), we briefly review the standard Clos network.

4.1 The Clos network

A 3-stage Clos network [5], denoted by $C(n, m, r)$, is an interconnection network where the first stage, made up of input switches, consists of r switches, each with n inlets and m uplinks. Symmetrically, the third stage consists of r output switches, each with n outlets and m downlinks. The second stage then is m switches, each with r links to first-stage switches and r links to third-stage switches. We call the switches in the middle stage the *core switches*. We refer to the links from a stage to a higher stage as *uplinks* and the links from a stage to a lower stage as *downlinks*. A *folded Clos network* places input and output layers top of each other, which we use in this paper, and when doing so, the input and output switches are the same devices, so we refer to them as input/output (IO) switches.

The recursive nature of Clos network means that we only have to deal with 3-stage Clos networks. An l-stage Clos network is recursively composed of 3-stage Clos networks. In an l-stage Clos network, each input and output switch is replaced by an $(l-2)$-stage network. An example is shown in Figure 2(a). As a result, any algorithm or theorem that applies to a 3-stage Clos network applies to an l-stage Clos

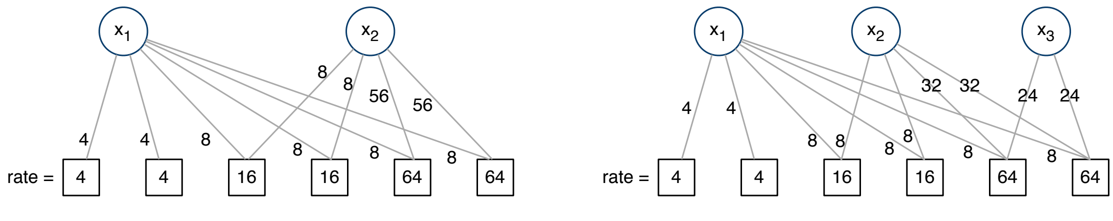


Figure 3: Two logical topologies for IO nodes with uplink rates $\{4, 4, 16, 16, 64, 64\}$). Despite the different number of root nodes, each of these topologies is optimal. There is also an optimal logical topology for these IO nodes with a single root node (as implied by Lemma 1).

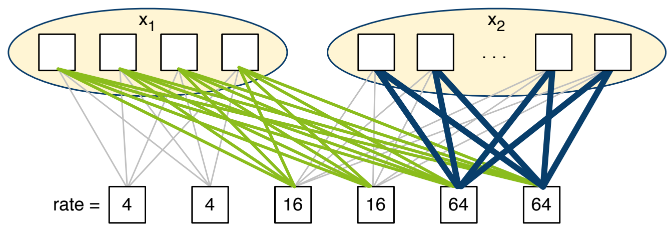


Figure 4: The physical realization of the logical topology shown in Fig. 3(a). Here, the logical root x_2 is realized by 8 switches (not all drawn for clarity), the thin links are unit capacity, the medium, green links are a bundle of 2 unit capacity links, and the thick, blue links represent 7 unit capacity links.

networks by applying it to the outermost 3-stage network first, and then recursively applying it to the $(l-2)$-stage subnetworks. As such, we always deal with 3-stage networks in this paper, but our results can be generalized to an l-stage Clos networks in a straightforward manner.

4.2 Constructing heterogeneous Clos networks

We separate logical topology design (§4.2.1) from the problem of finding a physical realization (§4.2.2). A logical topology in this context is a forest of trees where the leaves of these trees are IO nodes and each root node represents a set of core switches. If a root node x represents m switches in the physical realization, then a logical edge (i, x) between x and an IO switch i represents m physical links—one from i to each of the m switches represented by x. For example, the logical topology of a Clos network has a single logical root that has each IO node as its child; this is illustrated in Figure 2(b). The capacity of a logical edge indicates the bandwidth its physical links need to sum to so that the network can feasibly route all hose TMs., e.g., the logical edge (i, x) described just above has capacity m.

The logical topology design problem is to find a suitable set of root nodes, the neighbors of each root node, and the capacity of the edges between IO nodes and root nodes. First, we show in Lemma 1 how to find the root nodes and the edges between IO nodes and roots such that the logical topology is optimal, i.e., it uses the minimal amount of link capacity necessary and sufficient to feasibly route the hose TMs possible given the rates of the IO nodes. Finally, we show in Theorem 2 how to assign capacities to the logical edges. A set of IO

Given a logical topology, we then need to find a set of switches that realize each of its root nodes. As the logical topology is a forest, we can consider each tree in it separately, so our approach here is to find the switches of each root node individually. Theorem 3 shows that we can do this in such a way that we use the same amount of link capacity as the lower bound for feasibly routing the hose TMs.

4.2.1 Logical design

We are concerned with the design of logical topologies that use the minimal link capacity necessary and sufficient to feasibly route all hose TMs (i.e., the logical topology is optimal), and we make the assumption that a physical network can be realized using the same amount of switching capacity as the logical topology. We lift this assumption in the next section when we show how to find such physical realizations.

To support heterogeneous ToR switches, we do not require that each switch has n inlets and outlets as required by the classic Clos construction. Instead, we let each IO switch i have a rate, denoted by $r(i)$, which is the sum of its downlink rates (e.g., in a homogeneous network, the rate of each IO switch is n). Each logical edge (i, x) between an IO node i and logical root x has a capacity $c(i, x)$, which is the sum of physical link rates that (i, x) represents. A logical topology has *optimal edge capacity* if the sum of edge capacities is equal to the sum of node rates.

We are now ready to give our logical design results, starting with a characterization of the roots and their neighbors in optimal logical topologies.

LEMMA 1. *Let T be a logical topology with IO nodes $I = \{1, \ldots, k\}$, and let $x_1, \ldots, x_l$ be the root nodes of T. Let X_p denote the set of IO nodes neighboring root node x_p such that $X_1 = I$ and $X_1 \supset \cdots \supset X_l$. Whenever all edges of T have positive capacity, we have that T feasibly routes all hose TMs with optimal edge capacity if, for all x_p, such that $2 \leq p \leq l$,*

$$r(i) > \sum_{j \in X_{p-1} - X_p} r(j) \text{ for all } i \in X_p$$

and $|X_l - X_{l-1}| \geq 2$.

PROOF. All proofs have been omitted due to space constraints. See the full version of this paper [6] for details. □

The following results are implied by this lemma:

- whenever $r(1) = \cdots = r(k)$, the optimal logical topology has a single root node, and
- no matter the rates of each IO node, a logical topology with a single root node is optimal, i.e., a logical topology can always use fewer root nodes than it's allowed by Lemma 1 and be optimal.

Two optimal logical topologies for a set of IO nodes are shown in Figure 3.

This lemma identifies the available logical topologies for a set I of IO nodes, but it does not determine the capacities of each logical edge. Our next result shows how capacity can be assigned to the logical edges of T to feasibly route all hose TMs. The intuition underlying this theorem is that the root x_p and its children (the IO nodes) form a disjoint spanning tree. We provision the spanning tree rooted at x_1 first, and then move to the next root node's spanning tree. Every unit of capacity that is provisioned to x_1 is a unit that does not have to be routed through $x_2, \ldots, x_l$, so we subtract off the previously allocated capacity from the edges to $x_2, \ldots, x_l$.

THEOREM 2. *Let T, $x_1, \ldots, x_l$, $X_1, \ldots, X_l$, and I be as in Lemma 1, and let $X_0 = \emptyset$ and $X_{l+1} = \emptyset$. We have that T can feasibly route all hose TMs using optimal capacity if and only if*

$$c(i, x_p) = \begin{cases} \sum_{j \in X_p - X_{p+1}} r(j) & \text{if } i \in X_{p+1}, \\ r(i) - \sum_{j \in I - X_p} r(j) & \text{otherwise} \end{cases}$$

for all $1 \leq p \leq l$ and all $i \in I$.

We give an example of an optimally provisioned logical topology in Figure 3(a). Note that for the IO nodes given in Figure 3(a), an optimal logical topology could have 1, 2 (as shown), or 3 root nodes. This theorem prescribes the amount of capacity needed in a logical topology, yet it is flexible in assignment of this capacity across logical edges.

4.2.2 Physically realizing a logical node

We now show how to find a physical realization of a logical node. Here, we are given a logical root and a set of IO nodes, and we want to find a set of switches that realizes the core node.

Each IO switch has a set of uplink ports, which may have multiple speeds. To simplify our presentation, we separate IO nodes with multiple uplink port speeds into separate switches, so that each IO switch has a single uplink port speed. This does not lead to a loss of generality because we can recombine the separated switches later. So, each IO switch i has a single uplink port speed, denoted by $p(i)$. We assume that an IO switch i has at least $\lceil r(i)/p(i) \rceil$ ports; otherwise, no realization that can feasibly route all hose TMs exists.

We realize logical root x with a set I of IO nodes as its children. We use X to denote the set of switches that make up logical node x. Let $m(i) = \lceil c(i,x)/p(i) \rceil$, where $c(i,x)$ is the capacity of the logical edge (i,x) as before. Here, $m(i)$ is the number of physical uplinks i has to x. We use $P(r)$ to denote the set of all switches of I with $p(i) = r$, and $I(x)$ denotes the set of IO nodes neighboring root x.

Now, we determine how many switches comprise X and how many ports each has. Let $m_{\min} = \min_{j \in I(x)} \{m(j)\}$. The core switches that realize x and the IO nodes $I(x)$ form a complete bipartite graph, so we have $|X| = m_{\min}$. Each core switch in X must have at least $m_{\min} \cdot |P(r)|$ ports with speed r, for each port speed r, and each $i \in I(x)$ has $\lceil m(i)/m_{\min} \rceil$ uplinks to each switch in X. An optimal physical realization of the logical topology in Figure 3(a) is drawn in Figure 4.

The following shows our construction is optimal.

THEOREM 3. *A physical realization G constructed as described above of a logical tree T with root node x and IO nodes I with $c(i,x)$ minimized according to Theorem 2 can feasibly route all hose TMs.*

Further, if $c(i,x)$ and $m(i)$ are evenly divisible by $p(i)$ and $m_{\min}$ respectively for all $i \in I$, then the amount of link capacity used by this physical realization matches the lower bound of any interconnection network that can feasibly route all hose TMs.

In the above theorem we claim that our construction needs only as much link capacity as any other interconnection network that can feasibly route all hose TMs. An *interconnection network* is a network where nodes with positive rate (i.e., $r(i) > 0$) never directly connect to other nodes with positive rate, that is, all nodes connect to switches. A corollary to a result of Zhang-Shen and McKeown [30] is that any switching network with node rates $r(1), \ldots, r(n)$ can feasibly route all hose TMs iff the total link capacity is at least $\sum_{1 \leq i \leq n} 2\, r(i)$. This bound is matched by, for example, a homogeneous 3-stage Clos network when all IO switch rates are equal. Our construction matches this bound without any restrictions on IO switch rates.

5. LEGUP DETAILS

We now describe the details of LEGUP's optimization engine. Recall that the optimization engine solves a maximization problem by performing a branch and bound exploration of the aggregation switches. In this section, we focus on the handling of complete solutions, i.e., the candidate solutions with enough aggregation ports to connect all ToR switches with at least a spanning tree. Given a complete solution $S = \{s_1, \ldots, s_k\}$, where each s_i represents a switch, LEGUP does the following:

1. Bounds the cost of S (§5.1).
2. If S's bound is lower than the best complete solution found so far, S is trimmed and it is not branched. Otherwise, the feasibility of S is determined by:
   - selecting a min-cost set of core switches (§5.2); and
   - finding a physical mapping of the aggregation switches to the data center's racks (§5.3).

3. If S is determined infeasible (due to a budget or physical model constraint violation), then it is trimmed. Otherwise, the performance of S is computed (§5.4), the best complete solution is updated, and S is branched.

We use w_a, w_f, and w_r to denote the weights a our performance metrics agility, flexibility, and reliability respectively, so the overall performance of a solution S is $p(S) = p_a w_a + p_f w_f + p_r w_r$ where p_a, p_f, and p_r have been normalized by their maximal values. We show how to find these maximal values in (§5.4). Throughout, whenever we use one of these, we assume it has been normalized.

5.1 Bounding a candidate solution

Our bounding function estimates each performance metric individually and then returns the weighted sum of the estimates. Because it is used to trim solutions and we are maximizing performance, it must overestimate the best possible solution in the candidate solution's subtree. Given a candidate solution S, we bound each metric of S is found as follows.

Agility and flexibility Agility and flexibility are coupled, so we bound them simultaneously, i.e., we bound $w_a b_a + w_f b_f$. We begin by finding the maximum agility the remaining budget allows, that is, we find b_a^{max} by first greedily adding the switch with the highest sum of port speeds to cost ratio of all the available switch types to S until the cost of S is over-budget (note that this makes use of any existing switches that are not included in S as they have no cost). Since this bound is an overestimate, we do not worry about actually being able to realize the topology, so we aggregate the bandwidth of switches in S, and we use $r(S)$ to denote their aggregate bandwidth, i.e., the sum of their port speeds.

We combine all levels of switches into single logical nodes, that is, we have one a core node, aggregation node, and ToR node, which form a path ToR to aggregation to core. To find b_a^{max}, we need to find the maximum possible agility of this logical topology. Let $r(\text{ToR})$, $r(\text{aggr})$ and $r(\text{core})$ be the bandwidths of the logical aggregation and core nodes respectively. We observe that $r(\text{aggr}) = 2/3\, r(S)$ and $r(\text{core}) = 1/3\, r(S)$ maximizes b_a^{max} (this is implied by Theorem 2). Moreover, we have

$$b_a = \min\left\{1, \frac{r(\text{core})}{r(\text{ToR})}, \frac{1/2\, r(\text{aggr})}{r(\text{ToR})}\right\}.$$

We now lower bound b_f, denoted by $b_f^{\min}$. We have $b_f^{\min} = 1/2 r(\text{aggr}) - r(\text{ToR})$. That is, $b_f^{\min}$ is equal to the amount of spare bandwidth the aggregation and core nodes can handle without decreasing agility.

The algorithm we use to maximize $w_a b_a + w_f b_f$ is given in Alg. 1. Briefly, we maximize the sum by attaching 1 unit of capacity at a time to either the core or the aggregation switches depending on which position decreases agility the least. We repeat this process until $w_a b_a + w_f b_f$ hits a maximal point, which is guaranteed to be globally optimal because it is the sum of two linear functions.

Algorithm 1 — Bound agility and flexibility.

Input: $r(\text{core}), r(\text{aggr}), r(\text{ToR}), b_a^{\max}$, and $b_f^{\min}$
Output: b_a, b_f

begin
$b_a = b_a^{\max}$
$b_f = b_f^{\min}$
$r(cToR) = 0$
until the following does not increase $w_a b_a + w_f b_f$ **do**
  if $r(\text{cToR}) < r(\text{ToR})$ then
    $r(\text{core}) = r(\text{core}) + 1$
    $r(\text{aggr}) = r(\text{aggr}) - 2$
    $r(\text{cToR}) = r(\text{cToR}) + 1$
  else
    $r(\text{aggr}) = r(\text{aggr}) - 1$
    $r(\text{ToR}) = r(\text{ToR}) + 1$
  $b_f = b_f + 1$
  $b_a = \min\{1, \frac{r(\text{core})}{r(\text{ToR})}, \frac{1/2r(\text{aggr})}{r(\text{ToR})}\}$
end

Reliability We make two observations that upper bound S's reliability. We have that b_r is at most:

- 1/2 the number of ports on any $s \in S$; and
- the number of open ports on any ToR switch.

We therefore set b_r to the maximum of these two values.

5.2 Finding a set of core switches

To find the min-cost core switches, we need to solve two sub-problems: finding an optimal logical topology (§5.2.1), and then finding the min-cost switches that realize that topology (§5.2.2).

5.2.1 Selecting a logical topology

Theorem 2 allows for a wide range of logical topologies that can optimally connect a set of aggregation switches. We observe, however, that a logical topology with k logical core nodes can always be made to have $k-1$ logical core nodes by *stacking* switches, that is, by combining multiple switches with l ports in total into a single switch with at least l ports. Moreover, if no physical realization of a logical topology with k core nodes exists, then there is no physical realization of a logical topology with $k-1$ core nodes. Therefore, we always maximize the number of logical core nodes in accordance with Lemma 1. We set the capacities of each logical edge such that they are minimized according to Theorem 2.

5.2.2 Realizing the logical topology

Once we have a logical topology, we need to realize each logical node. We sketch LEGUP's realization algorithm and omit details of handling modular switches due to space constraints. The first issue is to determine the ports each aggregation switches should use to connect to ToR switches and what ones should connect to core switches. Again, we should have 1/2 the switch's bandwidth point each direction. We find aggregation switch down ports (i.e., the ports that connect to ToR switches) by iterating through the ToR switches. At each ToR switch, we select one of its free ports to use as an uplink by selecting its free port with the highest speed such that there is a switch in S with an open port at the same speed or greater. When multiple such switches in S exist, we connect this ToR switch to the $s \in S$ with the most free capacity. We repeat this procedure until either 1/2 the capacity of each switch in S has been assigned to a ToR switch or until the uplink rate of each ToR switch is equal to its hose traffic rate.

By Theorem 3, the aggregation switches and logical topology dictate the number of core switches and the number and speeds of ports for each core switch. A core candidate solution is therefore infeasible if one of the logical nodes cannot be realized because no switch has enough ports of each rate required (e.g., the aggregation switches may dictate that each core switch has 145 10Gbps ports when the largest available switch has only 144 such ports). Assuming that realizing the logical topology T is feasible, let $x_1, \ldots, x_l$ be T's logical root nodes. The switches that realize each x_i are dictated by X_i, the aggregation switches that are x_i's children, so we realize each x_i with the min-cost switch that satisfies its port requirements. This switch assignment is easily found by comparing each x_i's requirements to the available switch types.

We can, however, potentially lower the cost of the core switches by stacking several switches into one physical switch, e.g., if x_i needs to be realized by five 24-port switches, it can also be realized by a single 120-port switch, potentially at a lower cost. This switch stacking problem can be reduced to a generalized cost, variable-sized bin packing problem, which can be approximated by an asymptotic polynomial-time approximation scheme [8]; however, their algorithm is complicated and still too slow for our purposes since it must be executed for every complete solution. Instead, we use the well-known best-fit heuristic [17] to solve stack core switches, which is known to perform well in practice.

Two issues arise when we stack core switches. First, it is possible to turn a feasible solution infeasible, e.g., after stacking switches, the resulting solution may violate a physical constraint, such as there may not be a rack that has enough free slots for the larger switch. Second, stacking core switches can decrease our reliability metric. Therefore, we save the original set of core switches. If either of these cases occurs, we revert back to the original set of core switches, and then continue.

5.3 Mapping aggregation switches to racks and ToR switches

Now that we have determined the set of switches that comprise the aggregation and core levels, we need to place them into racks and connect each ToR switch to aggregation switches. We assume that the core switches can be centrally located and that ToR switches are already placed, so we are only concerned with aggregation switches in this section.

Our mapping algorithm takes as input a set of aggregation switches, here this is S, and the data center model. If no model is given, then this stage is skipped. If a network blueprint is given but no data center model, then the mapping assigns each link a unit cost if it is new or modified. The mapping's goal is to minimize the cost of the physical layout of these aggregation switches subject to the rack, thermal, and power constraints of the data center model; here, cost is the length of cables needed to connect the ToR switches to aggregation switches. Even using Euclidean geometry setting and without our additional constraints, this problem is NP-hard as it can be reduced to a Steiner forest problem variant, see, for example [16]. Additional complications here are that the data center model may already have aggregation switches in place and we would like to use Manhattan distance instead of Euclidean because cables in most data centers run above the racks in trays. These trays run above the rows and cross between rows perpendicularly to the row.

We use a two-phase best-fit heuristic for mapping. The first phase matches aggregation switches to existing switches in the data center model, and the second stage finds a best-fit for all aggregation switches not placed in the first phase. To speed up the algorithm, we do some preprocessing. The preprocessing and mapping algorithm details are given in Algorithm 2.

Phase I of our mapping algorithm attempts to replace existing aggregation switches in the data center model with a close switch in S. We define closeness as follows for two switches s_1 and s_2. We have $closeness(s_1, s_2) = 0$ if s_1 does not have as many ports as s_2 for any speed, when ports are allowed to operate at any speed less than their line speed, and $closeness(s_1, s_2) = 1$ if s_1 has at least as many ports as s_2 for all speeds, again allowing s_1's ports to operate at less than their max speed (e.g., the closeness of a 24-port 10Gbps switch and a 24-port 1Gbps switch is 1).

5.4 Computing the performance of a solution

We now address how to compute each of our performance metrics.

Agility can be found in polynomial time for any network using linear programming [21]. Here, however, we can use a faster algorithm. Because we have constructed the network in accordance with Lemma 1, a node i with rate $r(i)$ must have at least $r(i)$ of uplink bandwidth to feasibly route all hose TMs (i.e., for agility to be 1). Specifically, if the uplink bandwidth of all i's uplink ports sums to u, then we have that the network's agility is at most $u/r(i)$. We can therefore determine the upper bound on agility imposed at each ToR and aggregation switch to find the network's agility.

In general, reliability can be determined using a standard min-cut algorithm. A heterogeneous Clos network's reliability is bounded by the number of uplinks from a ToR to its aggregation switches and an aggregation switches to its

Algorithm 2 — Mapping aggregation switches to racks.

Preprocessing
Input: data center model
Output: lists of racks

begin
for each rack **do**
  find the sizes of its contiguous free rack units, and
  the distance to the k nearest ToR switches

Separate the racks into lists $R[u]$ such that the largest
  contiguous free rack units of racks in $R[u]$ is u

Sort each list in increasing order of distance to k ToR switches
end

Mapping
Input: data center model M and S
Output: map $S \rightarrow$ racks

begin
// Phase I
for each switch $x \in S$ **do**
  for each aggregation switch $y \in M$ **do**
    find the closeness of x and y
$S' = \emptyset$
for $y \in M$ **do**
  Map the closest $x \in S$ to y
  $S' = S' \cup \{x\}$
// Phase II
for each switch $x \in S - S'$ **do**
  Map x to the first rack in $r \in R[x.U]$
  Update r's largest contiguous rack units, and move it to
    the appropriate list
end

core switches as observed earlier, so we can compute it more quickly.

Computing flexibility depends on the rule specified for attaching new devices to the network. In our implementation, we greedily attach devices to the open port that reduces agility the least. Computing flexibility is done by repeating this process until no more unit bandwidth devices can be attached without reducing agility below δ.

Finding the maximal value of each metric We need to scale each of our performance metrics to a [0,1] range to compare them. The maximal agility of any network is 1; however, we normalize flexibility and reliability by finding the maximal value of each metric given the budget and using this for normalization. These upper bounds are found using our bounding function on an empty candidate solution.

6. EVALUATION

We now evaluate LEGUP by comparing it to other methods of constructing data center networks. We describe the data center used for evaluation first (§6.1), and then describe alternative upgrade approaches (§6.2). Finally, we study the performance of these approaches with two scenarios: upgrading our data center (§6.3) and expanding it (§6.4).

ToR switches

Hose uplink rate	Uplinks (1, 10 Gbps)	No. switches
28	8, 2	50
40	8, 4	80
8	8, 0	40
2	2, 0	20

Aggregation switches

Chassis	**Line cards**	**Free LC slots**	**No. switches**
HP 5406zl	3x 24 1Gbps, 1x 2 10Gbps	2	1
HP 5406zl	4x 4 10Gbps	2	9

Table 1: Existing switches in the SCS data center. In the ToR switch table, the hose uplink rate is the maximum rate the switch would like to send/receive at; the uplink indicates the number of free ports the ToR has with bandwidth 1 or 10 Gbps; and no. switches indicates the number of such switches in our input network. In the aggregation switch table, the chassis of all existing aggregation switches is an HP 5406zl; however, the configuration of line cards (LCs) for some aggregation switches is different as shown in the line cards column.

6.1 Input

Data center model To test LEGUP on an existing data center, we have modeled the University of Waterloo's School of Computer Science (SCS) data center. The servers in this room run services such as web, email, file storage, backup, and many are used as compute machines by faculty and students. To make the upgrade problem more like that in a larger data center, we have increased the number of racks and servers in the data center by an order of magnitude. We scaled the network proportionally, keeping the characteristics of the network invariant. Our analysis of the SCS data center is based on this scaled version. While the SCS is a small data center, choosing to study it rather than a made up system allows us to model a real-world data center with real constraints rather than synthesizing a model based on what we believe larger data centers look like.

The scaled-up SCS data center has three rows made up of 205 racks housing a total of 7600 servers, 190 ToR switches, six aggregation switches, and two core routers. These racks are arranged into three rows. Row 1 has 85 racks and rows 2 & 3 each have 60 racks.

The SCS data center has grown organically over time and has never had a clean slate overhaul. As a result, the SCS data center is a typical small data center with problems such as the following:

Heterogeneous ToR and aggregation switches: Switches have a long lifespan in the SCS data center, so the ToR switches are not uniform. Aggregation switches are all HP 5406zl switches, though they do not have identical line cards. The details of the data center's existing switches are listed in Table 1.

Poor air handling: The data center has a single chiller and it's located at the end of the rows. Additionally, the hot and cold aisles are not isolated, resulting in less effective cooling. Because of this, hot-running equipment cannot be concentrated at the far end of the rows where it will not receive much cool air from the chiller. We model this by linearly decreasing the allowed amount of heat generated per rack

Switch model	Ports	Watts	Price ($)
Generic	24 1Gbps	100	250
	48 1Gbps	150	1,500
	48 1Gbps, 4 10Gbps	235	5,000
	24 10Gbps	300	6,000
	48 10Gbps	600	10,000
	144 10Gbps	5000	75,000
HP 5406zl chassis	n/a	166	2,299
HP line card	24 1Gbps	160	2,669
HP line card	4 10Gbps	48	3,700

Table 2: Switches used as input in our evaluation. Prices are street and power draw estimates are based on a typical switch of the type for the generic models or manufacturers estimates, except for the HP 5400 line cards, which are estimates based on the typical power draw for a 1Gbps or 10Gbps port.

as the racks move away from the chiller. We do not have thermal measurements for all our input switches, so we approximate the thermal output of a switch by its power consumption. Full details of our model are in [6].

The data center's current network is arranged as a tree; each ToR switch has a single uplink to an aggregation switch and each aggregation switch has two uplinks to the core routers. We would like to modify the network so that only outbound traffic passes through the core routers. Therefore, all network upgrades must be three-levels, that is, they need to replace these routers with core switches.

Switch and cabling prices The switches available for use by the upgrade approaches are shown in Table 2. We assume that installing or moving links to or from an aggregation switch costs $50; links from ToR switches to servers are free to move. Based on our discussions with the data center operators, we believe this is a conservative estimate based on the price of cabling and the man-hours needed to install a cable in an existing data center. Though LEGUP supports charging for a cable based on its length, we do not use this functionality because we are currently unable to estimate the lengths of cables used by the fat-tree upgrade approach.

6.2 Alternative upgrade approaches

To evaluate the solutions found by LEGUP, we consider two alternative network upgrade approaches. The first method, is a naive scale-out algorithm. This algorithm upgrades line cards in existing modular switches as the budget allows by purchasing the line card with the least cost to rate ratio. When there are no more free line card slots, more HP 5406zl switches are purchased and filled with additional line cards in the same greedy fashion. This algorithm keeps the core and aggregation levels homogeneous and uses only 10Gbps links between the aggregation switches and the core switches (but uses heterogeneous links to the ToR switches since they have heterogeneous uplink rates).

The second approach we consider is to build a fat-tree using 1Gbps links following Leiserson's construction [23]; the fat-tree was suggested as a low-cost DCN topology by

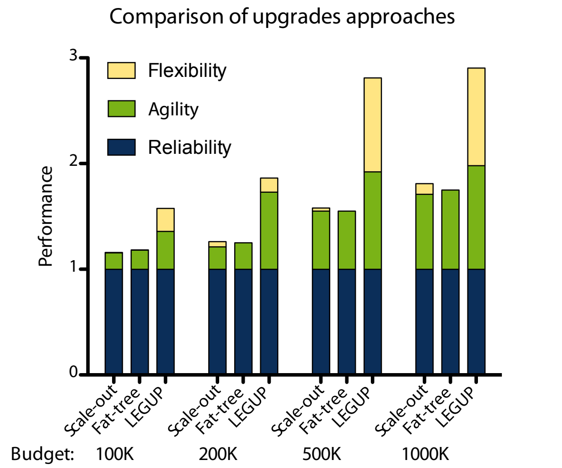


Figure 5: Performance of the upgrade approaches for various budgets. Here, we have $w_a = w_f = w_r = 1$ and $\delta = 0.10$.

Al-Fares *et al.* [2]. Here, we reuse existing ToR switches. We deviate from Al-Fares *et al.*'s strict definition of a fat-tree by allowing switches in different levels to have different radices, e.g., the aggregation switches could have 24 ports and the core switch could have 48 ports. This slight support for heterogeneity greatly improves the results of the networks found in our examples, and is supported under Clos's original network construction [5] (which is a generalization of the fat-tree).

For both these approaches we do not take the physical constraints of the data center into account. Therefore, it may not always be possible to construct the networks found this way. In contrast, LEGUP takes the physical constraints (in our case thermal and rack space) into account, and so it is at a disadvantage because its solutions are realizable.

6.3 Upgrading the data center

We first consider upgrading the SCS data center to maximize its performance. For this scenario, we set the weights of each performance metric to be 1 and $\delta = 0.1$.

The performance achieved by our three upgrade approaches for various budgets is shown in Figure 5. As the chart shows, for all budgets, LEGUP finds an upgrade with higher agility and flexibility than the the scale-out or fat-tree approaches. Moreover, LEGUP always finds a network upgrade with more agility and flexibility than the other two approaches even when LEGUP's budget is half as much as their budgets. Because the maximal reliability is two (as limited by the ToR switches with only two uplink ports), all upgrades were able to achieve this for all budgets.

Interestingly, the naive scale-out approach often outperforms the fat-tree. This is largely due to the high number of cables in the fat-tree, each of which costs $50 to install here. For example, with a budget of $100K, the fat-tree approach can only spend roughly $30,000 on switches because $70,000 is needed for cabling. By taking advantage of 10Gbps links, LEGUP and the scale-out approach need an order of

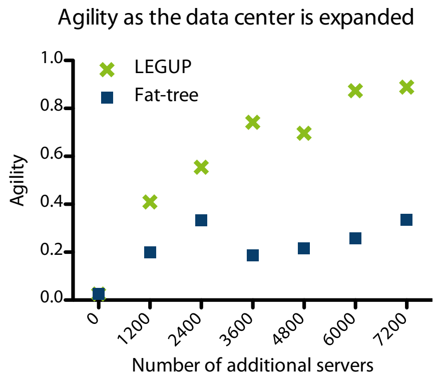


Figure 6: Agility as additional racks of servers are added to the data center. Each point is found by increasing agility as much as possible given a budget of $300,000 and the previous iteration as the existing network.

magnitude fewer cables.

6.4 Expanding the data center

We now consider expanding a data center network to accommodate additional servers as they are added over time. Again, we use the SCS data center as a starting point, and we add 1200 servers to it at a time and find a network for the expanded data center. Each expansion has a budget of $300,000, and uses the network found in the previous iteration as input. This budget was selected because it is 10% of the cost of the servers, assuming a price of $2500 per server; this cost is in line with recent cost breakdowns for servers compared to the network [11, 14]. We do not take the racks' thermal or constraints into account here because the assumption is that the data center floor would have to be expanded for any upgrade of this size. For LEGUP, we set $w_a = 1, w_f = 5, w_r = 1$ and $\delta = 0.10$. Because LEGUP assumes that servers connect to a ToR switches, we use 30 switches with 48 1Gbps and 4 10Gbps ports as ToR switches for each 1200 server expansion. Doing so uses $150,000 of LEGUP's budget each iteration.

The results of our expansion scenario are shown in Figure 6. LEGUP significantly outperforms the fat-tree upgrades. The fat-tree approach experiences a drop in agility when the network with 2400 additional servers is expanded by another 1200 servers because the aggregation and core switches of the +2400 server network are all 24-port switches. To accommodate the additional 1200 servers without lowering agility even further, its core switches need to be replaced by 48-port switches. After this change the amount of agility gained with each addition is less than previously because the 48-port switches are not as good a value as the 24-port switches.

7. DISCUSSION

Lacking a theoretical foundation to model and analyze heterogeneous tree-like topologies, a data center manager has two options to upgrade their network: (1) perform an expensive forklift upgrade, or (2) add additional switches to their network using best practices or other rules of thumb. This second approach would likely either result in a topology with sub-optimal agility for the money because link capacity would not be able to be used optimally. So, even without LEGUP, our theory of heterogeneous Clos networks is useful because it describes topologies that can extract maximal agility from available link capacity, which is useful to guide the addition of switches.

Moreover, our aggregation switch mapping algorithm given in Sec. 5.3 solves two challenges that arise even when using a traditional homogeneous network: where should aggregation switches be located? And, how to wire ToR and aggregation switches together to minimize cabling costs? We plan to fully evaluate this algorithm in future work.

And finally, LEGUP is a flexible optimization framework that can be expanded beyond the scope we have described here. For example, it is simple to extend LEGUP to account for switches with oversubscribed backplanes. This can be done by considering the impact of such switches on agility. It is also possible to account for other optimization metrics, such as end-to-end latency, in addition to agility, flexibility, and reliability. Doing so requires modifications to the bounding function (§5.1), however, and so we leave it to future work.

So far, we have not addressed operational issues that arise when heterogeneity is added to a DCN. We address them now:

- **Configuration:** We have not accounted for the cost of reconfiguring a DCN after modifying its topology. Reconfiguration could be expensive and error-prone, especially if it is performed manually. We expect that this will become less of a issue as data center management solutions improve. For instance, PortLand [25] provides "plug-and-play" functionality for DCN switches and NOX can be used to centrally manage a DCN [28]. Both of these solutions can support heterogeneous Clos topologies with minor modifications. As we mentioned above, LEGUP can be extended—we expect that reconfiguration costs could be added to LEGUP similarly to how rewiring costs are dealt with.

- **Routing and load balancing:** To prove the optimality of our heterogeneous Clos construction (§4), we assumed ideal load balancing. This is not achievable in practice because it requires support for splitting individual flows across multiple paths. Nevertheless, close to optimal load balancing on our constructions can be achieved, however. Mudigonda *et al.*'s SPAIN [24] performs multipath load balancing on arbitrary topologies. Based on their results, we believe SPAIN would be able to extract close to the full bisection bandwidth from our topologies. We believe that Multipath TCP [9] can also extract the full bisection bandwidth from our constructions based on early results [26].

8. RELATED WORK

Topology constructions Theoretical topology constructions date back to telephone switching networks and a variety of constructions have been proposed, e.g., Clos [5], Beneš [3], flattened butterfly [20], HyperX [1], hypercube [29], DCell [13], and BCube [12]. Despite the many topology proposals, the only other construction we are aware of that handles heterogeneous switches is that of Rasala and Wilfong [27], who gave a strictly nonblocking construction for networks with heterogeneous IO switches. Our work differs in two key aspects. First, they dealt with strictly nonblocking networks whereas our constructions are rearrangeably nonblocking in their setting (equivalent to feasibly routing all hose TMs in our traffic model), so our construction requires much less link capacity. Second, their constructions only connect IO switch sets with two types of switches and they do not support heterogeneous switch port speeds, whereas our construction supports any number of switch types and port speeds.

Network design The network design literature is vast, and algorithms for network design have been widely studied, see, e.g., [19]. Branch and bound has been used to solve network design problems in past work, for example, [15, 22]. However, existing work does not take the unique constraints of a data center environment into account. Our work here is the first to simultaneously support power, thermal, and rack space constraints. Moreover, other network design algorithms return an arbitrary mesh network. LEGUP returns only tree-like algorithms so that existing DCN addressing solutions can be used.

9. CONCLUSIONS

We have shown that heterogeneity can yield significant cost savings when upgrading or expanding a legacy data center. By developing the theory of heterogeneous Clos networks, we have given data center managers solid theory to rely on when designing upgrades. LEGUP performs this network design process by solving a difficult optimization problem with many constraints. LEGUP finds higher performing upgrades than existing solutions for less than half the cost. When incrementally expanding a network, LEGUP finds a network with 265% more agility than an upgraded fat-tree after the number of servers in the data center has been doubled.

10. REFERENCES

[1] J. H. Ahn, N. Binkert, A. Davis, M. McLaren, and R. S. Schreiber. Hyperx: topology, routing, and packaging of efficient large-scale networks. In *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis (SC '09)*, 2009.

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

[3] V. E. Beneš. *Mathematical Theory of Connecting Networks and Telephone Traffics*. Academic Press, 1965.

[4] T. Benson, A. Anand, A. Akella, and M. Zhang. Understanding data center traffic characteristics. In *Proceedings of the 1st ACM workshop on Research on enterprise networking (WREN)*, 2009.

[5] C. Clos. A study of non-blocking switching networks. *Bell System Technical Journal*, 32(5):406–424, 1953.

[6] A. R. Curtis, S. Keshav, and A. López-Ortiz. LEGUP: Using heterogeneity to reduce the cost of data center network upgrades. Technical report, University of Waterloo, 2010.

[7] N. G. Duffield, P. Goyal, A. Greenberg, P. Mishra, K. K. Ramakrishnan, and J. E. van der Merive. A flexible model for resource management in virtual private networks. In *SIGCOMM*, 1999.

[8] L. Epstein and A. Levin. An APTAS for generalized cost variable-sized bin packing. *SIAM J. Comput.*, 38(1):411–428, 2008.

[9] A. Ford, C. Raiciu, M. Handley, and S. Barre. TCP extensions for multipath operation with multiple addresses. *IETF*, 2009.

[10] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. Maltz, P. Patel, and S. Sengupta. VL2: a scalable and flexible data center network. In *SIGCOMM*, 2009.

[11] A. G. Greenberg, J. R. Hamilton, D. A. Maltz, and P. Patel. The cost of a cloud: research problems in data center networks. *Computer Communication Review*, 39(1):68–73, 2009.

[12] C. Guo, G. Lu, D. Li, H. Wu, X. Zhang, Y. Shi, C. Tian, Y. Zhang, and S. Lu. BCube: a high performance, server-centric network architecture for modular data centers. In *SIGCOMM*, 2009.

[13] 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 *SIGCOMM*, 2008.

[14] U. Hoelzle and L. A. Barroso. *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines*. Morgan and Claypool Publishers, 2009.

[15] K. Holmberg and D. Yuan. A lagrangian heuristic based branch-and-bound approach for the capacitated network design problem. *Operations Research*, 48(3):461–481, 2000.

[16] F. K. Hwang and D. S. Richards. Steiner tree problems. *Networks*, 22(1):55–89, 1992.

[17] D. S. Johnson, A. Demers, J. D. Ullman, M. R. Garey, and R. L. Graham. Worst-case performance bounds for simple one-dimensional packing algorithms. *SIAM J. on Comput.*, 3(4):299–325, 1974.

[18] S. Kandula, S. Sengupta, A. Greenberg, and P. Patel. The nature of datacenter traffic: Measurements & analysis. In *IMC*, 2009.

[19] A. Kershenbaum. *Telecommunications network design algorithms*. McGraw-Hill, 1993.

[20] J. Kim, W. J. Dally, and D. Abts. Flattened butterfly: a cost-efficient topology for high-radix networks. *SIGARCH Comput. Archit. News*, 35(2), 2007.

[21] M. Kodialam, T. V. Lakshman, and S. Sengupta. Maximum throughput routing of traffic in the hose model. In *Infocom*, 2006.

[22] E. L. Lawler and D. E. Wood. Branch-and-bound methods: A survey. *Operations Research*, 14(4):699–719, 1966.

[23] C. E. Leiserson. Fat-trees: universal networks for hardware-efficient supercomputing. *IEEE Trans. Comput.*, 34(10):892–901, 1985.

[24] J. Mudigonda, P. Yalagandula, M. Al-Fares, and J. C. Mogul. SPAIN: COTS data-center ethernet for multipathing over arbitrary topologies. In *NSDI*, 2010.

[25] R. N. Mysore, A. Pamboris, N. Farrington, N. Huang, P. Miri, S. Radhakrishnan, and V. Subram. Portland: A scalable fault-tolerant layer 2 data center network fabric. In *SIGCOMM*, 2009.

[26] C. Raiciu, C. Pluntke, S. Barre, A. Greenhalgh, D. Wischik, and M. Handley. Data center networking with multipath TCP. In *Hotnets*, 2010.

[27] A. Rasala and G. Wilfong. Strictly non-blocking WDM cross-connects for heterogeneous networks. In *STOC*, 2000.

[28] A. Tavakoli, M. Casado, T. Koponen, and S. Shenker. Applying NOX to the datacenter. In *HotNets-VIII*, 2009.

[29] H. Wu, G. Lu, D. Li, C. Guo, and Y. Zhang. MDCube: a high performance network structure for modular data center interconnection. In *CoNEXT*, 2009.

[30] R. Zhang-Shen and N. McKeown. Designing a predictable internet backbone with Valiant load-balancing. In *Thirteenth International Workshop on Quality of Service (IWQoS '05)*, 2005.