Taming the Flying Cable Monster: A Topology Design and Optimization Framework for Data-Center Networks

Jayaram Mudigonda* Jayaram.Mudigonda@hp.com
Praveen Yalagandula* Praveen.Yalagandula@hp.com
Jeffrey C. Mogul* Jeff.Mogul@hp.com
**HP Labs, Palo Alto, CA 94304*

Abstract

Data-center network designers now have many choices for high-bandwidth, multi-path network topologies. Some of these topologies also allow the designer considerable freedom to set parameters (for example, the number of ports on a switch, link bandwidths, or switch-to-switch wiring patterns) at design time. This freedom of choice, however, requires the designer to balance among bandwidth, latency, reliability, parts cost, and other real-world details.

Designers need help in exploring this design space, and especially in finding optimal network designs. We describe the specific challenges that designers face, and we present *Perseus*, a framework for quickly guiding a network designer to a small set of candidate designs. We identify several optimization problems that can be accommodated within this framework, and present solution algorithms for many of these problems.

1 Introduction

Imagine that you have been given the task to design a shipping-container ("pod") cluster containing 1000 server-class computers. The container provides the necessary power and cooling, you have already chosen the servers, and now you must choose a network to connect the servers within the pod.

Suppose that you know the pod will be used for large Hadoop (MapReduce-style) computations, and so application performance will depend on achieving high bisection network bandwidth within the pod. Therefore, you would like to use a multipath-optimized network topology, such as FatTree [4] or HyperX [3].

Each of the basic topologies has numerous parameters, such as the number of ports per switch, and you also must consider the cost of the network: switches, cables, connectors, and the labor to install it. So, how do you decide which topology and parameters are best for your pod? The design space, as we will show, can be complex and not always intuitive.

Now consider the problem from the point of view of a server-pod vendor: how do we decide what network designs to offer customers who expect us to ship them a container with a pre-wired network? The vendor would like to meet the needs of many different kinds of customers (e.g., Hadoop, scientific computing, Web hosting, Cloud computing) without having to support too many different designs, and without having to stock too many different kinds of parts.

In this paper, we expose the challenges of the network design problem for pod clusters, as well as other (non-containerized) data-center networks, and we describe a framework, called *Perseus*,[1] to assist the network designer in making the best choices, given a set of constraints. Our goal is to quickly quantify the costs and benefits of a large set of possible designs, and thus significantly reduce the search space. (Speedy design really does matter; good engineers are overworked.)

While other investigators have tried to analyze the relative merits of topology families (e.g., FatTree, HyperX, or BCube) [1, 13], we know of no prior work that describes how to efficiently search the design space for a given family.

The overall network-design problem gives rise to a variety of optimization sub-problems, many of which we will describe in this paper. Our framework allows us to incorporate optimizers for these problems.

Our contributions in this paper include:

- An overall characterization of the problem of topology optimization for data-center networks on the scale of a single container, or similar-sized clusters.
- A description of the workflow for designing a cost-effective network, given specified performance targets and parts costs.
- Description and solution of several interesting optimization problems, which (to our knowledge) have not previously been discussed in the literature.
- Results, based on semi-plausible parts costs, showing that overall network cost depends on topology parameters in ways that are sometimes hard to predict, reinforcing the need for a tool to explore the design space.

1.1 Problems this paper does not address

The topic of exploring the topology space includes several important problems, such as practical solutions to the flow-routing problem within multipath networks,

[1]Perseus slew the snake-haired Medusa.

or accurate estimates of network costs, or comparing the general merits of HyperX vs. FatTree networks, *that we do not intend to solve in this paper*.

The actual bandwidth attainable in a multipath network depends on how flows are routed, and is likely to be less than the network's bisection bandwidth. Various papers have described computationally-feasible methods for routing in such networks [3, 5]. However, without a realistic traffic model, it is hard to choose routes or predict actual bandwidths; we therefore follow the lead of others (e.g., [13]) in using bisection bandwidth as a crude approximation of a network's useful capacity.

We developed our framework in response to a request from product architects and engineers, who must make cost vs. performance tradeoffs. The engineers we work with have accurate volume-cost estimates for parts, *which we cannot use in this paper, because these costs are subject to non-disclosure agreements*. In this paper, so as to provide an *illustration* of the results of our approach, we use Web-retail prices instead of the true parts costs. Further, such retail prices would be a meaningless basis for comparing between (say) HyperX and FatTree topologies for a 1K-server cluster, because the ratios between these prices do not reflect the ratios between actual costs; therefore, we do not attempt to make such comparisons. (Popa *et al.* [13] have made the attempt.)

We also note that the choice between topology families (FatTree, HyperX, BCube, etc.) depends on other considerations beyond bandwidth and parts costs – e.g., which topologies best support energy proportionality at data-center scale [1], or whether a server-based topology such as BCube is acceptable to customers. This paper does not address those considerations.

Although there are similar design constraints for the network that connects multiple pods into a data-center-wide network, quantitatively this is a significantly different problem, and appears to require qualitatively different networking solutions (e.g., hybrid electrical/optical switch architectures [7]). Therefore, we do not currently attempt to extend our approach to this scale.

2 Defining the problem

Our goal is to help a data-center network designer in a world with too many choices, both for how to design a network, and for how to evaluate a set of designs.

2.1 Points of view

We know of several different types of network designers, with somewhat different points of view: large-scale data-center operators, and external design consultants, with sufficient expertise to design their own networks; moderate-scale data-center operators who prefer to stick to well-understood choices; and system vendors.

Our goal is to serve all of these points of view, although the full flexibility of our approach may be of most interest to a system vendor, who has the most ability to choose the low-level parts. System vendors are increasingly asked, by large customers, to do rack-scale or container-scale pre-integration of servers and networks. Vendor designers have a lot of freedom in terms of the parts they can use; however, they are constrained by the need to choose designs that apply to a large number of customers.

Our work on this problem was, in fact, inspired by a request from pod system designers, who need to know how to design rack-level switches that will be useful for a wide variety of customers (and thus must evaluate network design tradeoffs long before knowing the traffic matrix for any specific customers).

2.2 Design choices

At an abstract level, a data-center network is composed of hosts, switches, and links. We focus our attention on flat Layer-2 designs; the use of IP subnets within data centers complicates some of the design choices and is worth further work.

Recent papers have described a variety of scalable, multi-path Ethernet-based data-center network topologies. These designs use a regular topology (we discuss topologies in Sec. 3) and are typically intended to exploit low-cost, "merchant silicon"-based Ethernet switches.

Generally, these topologies have several free parameters, including the number of end-hosts ("terminals") attached to edge switches (in some designs, all switches are edge switches), the number of ports per switch (the switch "radix"), and link speeds, both of which can vary even within a single network. Switch port count, link speeds, and table sizes all affect overall system cost; we defer detailed discussion of costs until Sec. 4.

For example, some existing merchant-silicon switch products support at least 64 10GbE ports; we expect more and faster ports in the relatively near future.

Each host may have one or more NICs, with per-NIC link speeds of 10 Gbps or even higher. In this paper, we generally ignore the details of end-hosts, including issues such as host virtualization (and therefore the use of virtual switches.)

At a level below the abstraction of switches and links, the network designer must consider more mundane issues, such as cables and connectors. These issues can significantly affect costs (see Sec. 4).

The designer must also consider the physical layout of the equipment. We assume that we will need multiple switches per rack (since most inexpensive switches do not have enough ports for all of the server-NIC downlinks in a rack, as well as for all of the necessary switch-to-switch links in a modern topology). Even so, as we will show, the designer then faces a choice between packing the racks with as many servers as will fit, or avoiding cables that connect a server in one rack with a switch in another rack.

2.3 Design constraints

Network designers face not just choices, but constraints. These can include:

- **number of connectors**: connectors take up area on the faceplates of switches; as cabling complexity increases, this can become a limiting factor.
- **copper cable length**: Copper cables at multi-gigabit speeds have practical length limits, typically 5–10 meters or less.

There are other similar issues (e.g., the weight of copper cables, the finite capacity of cable plenums, the propensity of massive cable bundles to block airflow and complicate cooling, and limits on the bending radius of all kinds of cables) that we have not yet considered. In addition, switch power consumption is both a major concern and beyond our current ability to model; see [1] for relevant work.

The algorithms that we are developing for Perseus can handle some of these constraints; we currently defer others to future work.

2.4 Evaluation metrics

Given a set of candidate network designs, we must compare them on some bases. There are usually numerous metrics by which to compare two designs, and they seldom agree. Rather than trying to find a single optimal design, our approach is to narrow the design space and then to expose the relative merits of the alternatives to a human designer.

We focus on two primary performance metrics: bandwidth and latency.

We use "bisection bandwidth" as a way to measure the capacity of an entire network design.[2] Following Dally [6], we define "bisection bandwidth" as the "minimum bandwidth over all bisections of the network." In this paper, we do not consider any particular multipath routing scheme, but instead assume that whatever routing scheme is chosen can fully utilize the network hardware and cables.

For reasons of cost, designers must often design networks with non-uniform oversubscription (NUO). Our topology generators can easily create NUO networks, but optimizing these designs and visualizing the results requires us to define a scalar performance metric other than network-wide bisection bandwidth. This is a straightforward change to our algorithms (although it can invalidate some of our heuristics, such as H3 in Sec. 6.3), but space does not permit us to discuss the various possible metrics or how they change the results.

[2]Sometimes we use the related term "oversubscription ratio."

We approximate latency in terms of hop count. Ignoring the considerable latency in end-host stacks, switch hops are still the primary source of delay in an *uncongested* data-center network, and is of great importance to applications such as scientific and financial computing. Characterizing the actual per-switch delay is beyond the scope of this paper.

In addition to performance, a network designer must also consider other metrics, including reliability and power. We discuss some prior work on power in Sec. 9.

Note that the topologies we discuss are all multi-path and hence inherently redundant. One could quantify reliability in terms of the vulnerability of the network to a certain number of randomly-chosen link and/or switch failures, but we are not aware of prior work that describes data-center network failure rates. Some researchers have described their designs as fault-tolerant, but (for example) Mysore *et al.* [11] discuss the reconvergence time of PortLand, but do not quantify its underlying vulnerability. However, Guo *et al.* [9] do show how the aggregate throughput of several topologies, including FatTree and their own BCube design, degrade under random failures.

We believe that it would be quite interesting to understand how to simultaneously optimize the tradeoff between bandwidth, latency, fault tolerance, energy, and cost – but this is beyond our current abilities.

3 Multipath topologies

In recent years, researchers have proposed a wide variety of topologies for data-center networks, all with the goal of providing high bisection bandwidth at low cost. Most of these topologies also require choices to be made for a variety of parameters.

Table 1: Symbols used in this paper

N	total number of servers (or external connections)
R	switch radix (port count)
T	terminals connected to a single switch
S	total number of switches
L	levels of a tree
D	dimensions in a HyperX network
K	link bandwidth
W	total number of links in a network
C	number of top switches in a tree

In this paper, we consider these topology families:

- **FatTree**: Rather than limiting our approach to the three-level k-ary fat tree structures described by Al Fares *et al.* [4], we consider a generalized version of the Clos topologies with parametrized *levels* and *fatness* at each level, which were first defined by Öhring *et al.* [12] as Extended Generalized Fat Trees (EGFTs).

  We recursively define an L level EGFT as follows: A level $L = l$ EGFT connects M_l of $L = l - 1$ level EGFTs with C_l top switches; each top switch has a K_l-wide connection to each of the $l-1$ level EGFTs. (I.e., K_l is the Link Aggregation (LAG) factor.) A level $L =$

1 EGFT has just one switch ($C_1 = 1$), with M_1 servers directly connected to the switch with $K_1 = 1$ (i.e., unit-bandwidth) links. Note that the level-1 EGFT can be generalized further to consider servers with multiple interfaces. We represent an EGFT as EGFT($L, \vec{M}, \vec{C}, \vec{K}$) where $\vec{M}, \vec{C}, \vec{K}$ are vectors of size L.

Properties: The total number of switches S, number of nodes N, and number of links W in a EGFT($L, \vec{M}, \vec{C}, \vec{K}$) can be computed as follows:

$$\begin{aligned} S &= C_L + M_L(C_{L-1} + \\ &\quad M_{L-1}(...(C_3 + M_3(C_2 + M_2))...)) \\ N &= \prod_{l=1}^{L} M_l \\ W &= C_L M_L K_L + M_L(C_{L-1} M_{L-1} K_{L-1} + \\ &\quad M_{L-1}(...(C_2 M_2 K_2 + M_2(M_1))...)) \end{aligned}$$

If all of the multiple links between two switches can be aggregated into a single cable, then the cable count W' will be:

$$\begin{aligned} W' &= C_L M_L + M_L(C_{L-1} M_{L-1} + \\ &\quad M_{L-1}(...(C_2 M_2 + M_2(M_1))...)) \end{aligned}$$

At a level $l \in [1, L]$ of EGFT($L, \vec{M}, \vec{C}, \vec{K}$), each of the C_l top switches provides $M_l K_l$ bandwidth to all the terminal servers in that sub-fattree. Hence, the oversubscription ratio at level l, referred to as O_l is

$$O_l = \frac{\prod_{i=1}^{l} M_i}{C_l M_l K_l} \quad (1)$$

The oversubscription ratio O of a EGFT($L, \vec{M}, \vec{C}, \vec{K}$) is $O = \max_{l=1}^{L} O_l$. The bisection bandwidth is N/O and the maximum number of hops between any two servers is $2L$.

- **HyperX**: HyperX [3] is an extension of the hypercube and flattened butterfly topologies. Switches are points in a D-dimensional integer lattice, with S_k switches in each dimension $k = 1..D$. The dimensions need not be equal. A switch connects to all other switches that share all but one of its coordinates. (E.g., in a 2-D HyperX, a switch connects to all switches in the same row and in the same column.) The link bandwidths $K_1, ..., K_D$ are assumed to be fixed in each dimension, but can vary across dimensions. At each switch, T ports are assigned to server downlinks.

  We can describe a network as HyperX($D, \vec{S}, \vec{K}, T$), with $\vec{S}$ and $\vec{K}$ as vectors. HyperX($D, \vec{S}, \vec{K}, T$) has $\prod_{k=1}^{D} S_k$ switches, $T. \prod_{k=1}^{D} S_k$ servers, and $(S/2). \sum_{k=1}^{D}[(S_k - 1).K_k]$ links.

In this paper we focus on EGFT and HyperX topologies because they are considered attractive for high bisection bandwidth data-center networks. However, we plan to support other interesting server-to-server topologies such as BCube [9] and CamCube [2], as well as traditional 2- or 3-tier topologies, to allow designers improved flexibility.

4 Cost model

In order to optimize the total cost of a network, we must have a cost model. Some costs are relatively easy to model; these include:

- **Parts costs**: These cover things that a system vendor would buy from other suppliers, such as switches, cables, and connectors.
- **Manufacturing costs**: Given the large physical size of a container-based cluster and the relatively small quantities manufactured, cables for these systems are installed by hand. Sec. 4.2.1 discusses this cost.

Other costs are harder for us to model, especially since they depend on factors beyond the costs to manufacture a specific cluster:

- **Design costs**: A network designer must spend considerable time understanding the requirements for a network, then generating and evaluating specific options. Our approach aims to reduce this cost, while improving the designs produced.

  A vendor of container-based clusters would prefer to deal with a limited number of designs, since each new design requires new Quality Assurance (QA) processes, and each new design must be explained to justifiably skeptical customers.
- **SKU costs**: When a system vendor must deal with a large variety of different parts (often called *Stock-Keeping Units* or SKUs), this creates complexity and generally increases costs. One of our goals, therefore, is to generate network designs that require only a small set of SKUs – generally this means only a few types and lengths of pre-built cables.
- **Cost to reconfigure the network**: Some clusters are born large; others have largeness thrust upon them, later on. A good network design allows for the incremental installation of capacity – for example, one rack of servers at a time – without requiring the re-wiring of the existing network. When such rewiring is required, it should be minimized.
- **Maintenance costs**: Electronics, cables, and connectors do fail. A network design that confuses repair people will lead to higher repair costs and/or more frequent mis-repairs.

In the current version of our framework, we model only the parts costs. Because system vendors and their suppliers are reluctant to reveal their actual volume-purchase parts costs, in this paper we instead use Web-published retail prices as proxies for real costs.

We do not claim that these are the costs that would be paid by a system vendor, but they serve to illustrate many of the important tradeoffs, and we can use them as a plausible input to our topology cost evaluation.

4.1 Switch costs

One can choose from a wide variety of switches for use in a data-center network. Switch costs vary tremendously based on feature set, port count, and performance.

For the sake of simplicity, based on a survey of various switch list prices, we will arbitrarily model switches at \$500 per 10GbE port, consistent with the value of \$450/port used by Popa *et al.* [13]. (Our tool can easily use table-based inputs for specific switch configurations if they are available.) We also assume that switches are available with exactly the mix of connectors that we would like to use, although these might not currently be off-the-shelf configurations.

We assume that all switches are non-blocking – that is, they do not impose a bandwidth limit beyond what is imposed by their port speeds.

Some researchers have considered the use of merchant-silicon Ethernet switch ASICs, along with the associated supporting parts (CPU, PHYs, power supply and fans, circuit board, etc.) to build low-cost special-purpose switches for data-center networks. This might also seem to be a way to model the cost of higher-radix switches (for example, the Broadcom BCM56840 supports 64 10GbE ports). Farrington *et al.* [8] analyzes the costs of one example of such an approach. However, it turns out to be extremely difficult to estimate the parts costs for such switches; prices for these ASICs are usually non-disclosure information, and it is tricky to estimate the cost of the additional parts (not to mention the cost of engineering, manufacturing, QA, and compliance with standards and regulations). Therefore, in this paper we do not attempt to estimate the costs of this approach, although one could guess that it might not be a lot lower until one is dealing with very large quantities of switches.

We note that some recent data-center network designs, such as CamCube [2], dispense entirely with discrete switch hardware. In such designs, all switching is done within a multi-port server/NIC complex – "each server connects directly to a small set of other services, without using any switches or routers" [2]. We believe our approach could easily be extended to model the costs of switchless networks, by setting the switch cost to zero and including appropriate costs for the required additional NIC ports. It might be harder to model the achievable bandwidth and delay in these networks, since the involvement of NICs or server CPUs at each hop could affect performance in complex ways.

4.2 Cabling costs

High-performance cables are expensive, and can easily amount to a significant fraction of the cost of the servers in a cluster. In Sec. 7.4, we will discuss the problem of optimizing the choice among a number of different cable options. In this section, we discuss cable-cost issues, based on published prices.

There are many options for cabling a high-performance network, and Perseus could easily be extended to cover them, but for this paper we have narrowed our focus to a few likely choices: copper or single-mode fiber, using SFP+ or QSFP+ connectors in either case, with 10GbE connectivity in all cases.[3]

For simplicity of both manufacturing and presentation in this paper, we will assume that any given network design uses only a single connector type. QSFP+ connectors have the benefit of working with either electrical or optical cables, which allows flexibility in cable choice without creating complexity in switch-configuration choice. Fiber QSFP+ cables have the electrical-optical conversion chips integral to the connectors, which adds cost but supports this flexibility. Therefore, we assume the use of single-channel SFP+ cables between servers and switches (and sometimes between switches, for short runs), and quad-channel QSFP+ cables for longer or wider paths between switches. [4]

Table 2: Cable prices (dollars) for various lengths

	Single channel	**Quad channel**		
Length (M)	SFP+ copper	QSFP copper	QSFP+ copper	QSFP+ optical
1	45	55	95	—
2	52	74	—	—
3	66	87	150	390
5	74	116	—	400
10	101	—	—	418
12	117	—	—	—
15	—	—	—	448
20	—	—	—	465
30	—	—	—	508
50	—	—	—	618
100	—	—	—	883

Sources: `http://www.cablesondemand.com/` and `http://www.elpeus.com/`

Table 2 shows cable prices, in dollars, for various lengths of copper and fiber cables certified to run at 10 Gbps. Although these are quantity-50 list prices, not the actual costs to a system vendor, for simplicity we will treat cable costs as equal to cable prices.

One implication of these costs is that a container-sized network might well need to use a mix of copper and optical cables to minimize total cost. This is because copper cables are significantly cheaper than optical cables of the same length; however, quad-channel copper cables cannot support 10GbE over more than 5 meters (SFP+ cables can support longer lengths because they use thicker cables, but consume more connector area as a result). [5] Above 5 meters, we must generally use optical cables.

[3]Popa *et al.* [13] advocate using CAT6 cables, which are cheaper but which require perhaps 3 times more power, and may be harder to use in dense wiring plans.

[4]Some cost-optimal configurations might "waste" channels in these quad-channel cables.

[5]"Active" QSFP copper cables can span 20 meters, but cost almost as much as fiber QSFP+ cables and so we do not consider them.

Thus, the cost-optimal network might depend on finding a topology that exploits short cables as much as possible, which in turn affects the optimization of the physical wiring plan (see Sec. 7.1). On the other hand, physical constraints might force the use of fiber cables even when copper cables would be short enough; see Sec. 7.4.

Based on the prices in Table 2, we can model quad-channel copper cables as costing ca. \$16 per meter, plus \$20 for each connector. Similarly, we can model quad-channel optical cables as costing ca. \$5/meter, plus \$188/connector. We model single-channel copper cables at \$6/meter, and \$20/connector. Custom lengths in small quantities will cost more than high-volume standard lengths, but we are offering these as only crude estimates of costs, and they will suffice for our purposes.

We cannot currently model the extra cost of using more than the minimal number of cable SKUs. Instead, our physical layout algorithm (see Sec.7.1) can either generate networks using custom-length cables or using only the standard cable lengths from Table 2, to illustrate the effect on SKU count, which we quantify in Sec. 8.1.

4.2.1 Cable installation costs

Independent of the parts cost for cables, we also must pay to install them. As far as we know, current manufacturing practices require manual installation of cables between switches.

We estimated cable-installation costs based on a recent experience in a 9-rack, 256-server cluster. A skilled installer can install about 20 intra-rack cables per hour, and about 8 inter-rack cables per hour, although the times depend a lot on cable length and distance between racks. (This experience also points out that the installation rate drops a lot if the installer must deal with cables that are cut too long – finding space for the excess cabling is tricky – so an accurate 3-D model for cable runs could be quite useful. Our algorithm in Sec. 7.1 attempts to model cable lengths as accurately as possible.)

In our area, skilled cable installers can charge ca. \$50/hour, so for this paper, we assume a cost of \$2.50 for installing an intra-rack cable, and \$6.25 for an inter-rack cable. These costs are significantly less than optical-cable costs, but they are not negligible when compared to copper-cable costs.[6] In the long run, we expect cable part costs to decline, but installation labor costs could rise unless there are process improvements that make installation quicker.

5 The topology planning workflow

We can now describe a workflow by which a network designer can explore the space of possible topologies. In brief, the steps of this workflow are: (1) The user specifies the problem, and chooses one or more basic topologies to compare; (2) Perseus generates acceptable candidate topologies, generates optimized wiring plans, estimates overall system cost, and generates a visualization of the resulting space. We describe each of these steps in more detail.

[6]Popa *et al.* [13] argue that labor costs dominate the costs of CAT6 cables.

5.1 User-specified inputs

The process of network design starts with a set of goals, constraints and other inputs, which must be provided by the user. These fall into several categories:

System parameters:

- Number of servers to support.
- Server NIC bandwidth (e.g., 1GbE or 10GbE).

While many systems include redundant NICs for fault tolerance, we will consider only non-redundant NICs in this paper. *While our tools can handle a variety of NIC and switch-port bandwidths, for simplicity we assume 10GbE throughout this paper.*

System goals:

- Desired minimum bisection bandwidth, internal to the cluster.
- Desired minimum outgoing bandwidth from the entire cluster. To simplify the rest of the process, we convert this into an equivalent number of “phantom” servers. For example, if the designer wants a 1024-server cluster with 10GbE NICs and 100 Gbps of external bandwidth, we design a network for $1024 + (100/10) = 1034$ server-equivalent ports. This gives the designer freedom to attach external connections to any switch in the cluster.

  This approach to external bandwidth creates some potential inaccuracy in our bandwidth calculations, as we discuss in Sec. 10.
- Desired maximum hop count.
- Desired maximum number of racks.

Parts available:

- Server size, in rack units.
- Switch types: a set of possible switch parts, with port counts and speeds (e.g., “48 1GbE ports + 4 10GbE ports”), and their sizes in rack units.
- Cable types (copper or fiber) and available lengths and bundling factors. We would also like to know a cable’s cross-sectional dimensions.

For each kind of part, the user must also supply a cost model. (See Sec. 4 for some example cost models.)

System constraints:

- The maximum number of uplink connectors per switch. (If cables are bundled, a single connector supports multiple logical links.)
- Rack height.
- Desired physical layout of racks: maximum row length, maximum number of rows, and width of aisles

between rows.

- Plenum cross-section dimensions.

The user must also decide on one or more of the base topology types that the tool supports (currently, just Fat-Tree and HyperX). This choice might depend on considerations that we cannot currently model, such as the expandability of a basic design, or the willingness of customers to accept it.

5.2 Generating candidate topologies

The Perseus tool starts by generating a set of candidate abstract topologies: those that meet the constraints and requirements provided by the user. (If no such candidates exist, the user will have to loosen the requirements and try again.)

This step varies depending on the base topology type, and we describe several appropriate algorithms in Sec. 6.

5.3 Converting abstract to physical wiring plans

After choosing a set of candidate topologies, we must then generate physically valid wiring plans before we can assign costs, since cable costs depend on their lengths. Sec. 7 describes this process in detail, including several topology-specific algorithms. Once we have chosen a physical topology for each candidate abstract topology, we can calculate cable lengths and types.

The least-cost wiring plan might use copper cables for shorter distances, since these could be cheaper than fiber cables of the same lengths. Because plenum space might be a concern, especially for copper cables, we then have to calculate whether the cables will fit. If not, we must replace copper with fiber until everything fits. See Sec. 7.4 for more details.

5.4 Visualization of results

Once we have a list of parts, including specific cable types and lengths, we can easily generate an estimate of the total system cost.

Given the large design space, it would be nice to have a clever multi-dimensional, navigable visualization of the space, including costs, bandwidths, and latencies for a large range of designs.

So far, however, we are using 2-D plots (e.g., network cost vs. bisection bandwidth), with curves for a small variety of parameter settings, as a way to approximate slices of a fancy visualization. Sec. 8 includes some examples. We also have a simple tool that shows how wires are laid out in 2-D space, but for any interesting-sized network, the pictures are too dense to fit into this paper.

6 Generating candidate topologies

In this section, we describe our algorithms for generating candidate topologies of several basic types.

Algorithm 1 Generate HyperX candidate topologies

```
1: Given:
2:    $N$: Number of servers in the pod
3:    $S$: Number of switches
4:    $R$: Number of ports per switch
5:    $T$: Number of terminals per switch
6: Goal:
7:    An HyperX config with the minimum cabling cost.
8: $R_h = R - T$ /* number of HyperX ports per switch */
9: if $R_h \leq 1$ then
10:    return $infeasible$ /* not enough ports */
11: /* Initialize the set of candidates with the 1-dim config */
12: $C = \{D = 1, S_1 = S\}$
13: while $C \neq \emptyset$ do
14:    $config = next\_config(C)$
15:
16:    if $R_h \geq (\sum_i (S_i) - D)$ then
17:       $assign\_Ks(config)$ /* see Sec. 6.2 */
18:       $output\_config\_details(config)$
19:
20:    $C = C \cup split\_dimensions(config)$
21: End
```

6.1 Generating HyperX candidate topologies

When designing an abstract HyperX topology, the designer must evaluate a potentially large space of candidates. Recall that the parameters that characterize this family of topologies are HyperX$(D, \vec{S}, \vec{K}, T)$, where D is the number of dimensions, $\vec{S}$ is the vector of number of switches along each dimension, $\vec{K}$ is the vector of link bandwidths along each dimension, and T is the number of terminals (servers) attached to each switch. (As noted in Sec. 5.1, we currently treat external bandwidth requirements as additional phantom servers.)

The goal of the algorithm described in this section is to generate all of the plausible candidate topologies (based on several constraints), which can then be ranked according to their costs. For the sake of simplicity, we assume that all NICs and server ports have the same unit bandwidth, and that all switches are identical and have the same number of servers attached. (In Sec. 6.1.1 we discuss relaxing the last constraint.)

We state the problem formally: Given N servers (or server-equivalents, to account for external bandwidth), and S switches of radix (port count) R, generate a set of the best feasible HyperX topologies.

Algorithm 1 shows our algorithm in pseudo-code. The first step simply derives the number of ports per switch available for intra-cluster links; we call these the "HyperX ports" to distinguish them from terminal (server and external) ports.

The algorithm then iterates over all possible dimensionalities (values for D) and adds the feasible candidates to the candidate set. For each iteration, we:

- Generate the candidate topologies for this dimensionality D. That is, we generate all possible values of $\vec{S}$ for D.

  For $D = 1$, the only candidate is a linear topology.

For each $D > 1$, we take each of the candidates for $D-1$ and, if possible, split one of its dimensions. For example, a 6x6 2-dimensional lattice can be split into a 6x3x2 lattice, and on the next iteration into a 3x3x2x2 lattice.

- Test the structural feasibility of the candidate; each switch must have enough HyperX ports to connect to all the remaining switches along each dimension.
- For a feasible candidate, find the optimal trunking factor (LAG factor) along each dimension – that is, we generate $\vec{K}$. The designer might prefer LAG factors that are a multiple of the connector and cable width (for example, QSFP connectors that support four-channel cables). A naive approach would require us to examine $\prod_{i=1}^{D}(R-T-i)$ (which, in case of a 5-dimensional HyperX with a 96-port switches, translates to about 3.8 billion) different configurations. Sec. 6.2 presents an $O(R)$ algorithm that derives the optimal trunking factors without exploring this huge space.

Note that when we split solutions from dimension $D-1$ in order to generate new candidates for dimension D, we must include as starting points all of the previous candidates, including the infeasible ones – the progeny of an infeasible candidate might themselves be feasible.

From the feasible candidates, we should then prune out those that require too many connectors to fit on a switch faceplate. We defer this step to future work, although it is fairly simple.

The complexity of Algorithm 1 is determined by the number of unique partitions of the set of prime factors of S; this can be very large, but the algorithm runs fast enough for practical values of S.

Once we have generated the entire set of feasible candidates, we can compute bisection bandwidths (using $\min(S_iK_i)(S/4)$ [3]) , maximum hop counts, and construction costs for each of these.

6.1.1 Optimizing HyperX by adding switches

In the description above, we generate a set of HyperX topologies that exactly match the specific number of switches S. We assume that the designer would like to minimize the number of switches, hence the choice of S. However, the construction in Alg. 1 finds only topologies with exactly the requested number of switches.

Sometimes, adding a few switches might enable much better configurations. For example, suppose the designer specifies a 31-switch network. Since 31 is prime, this forces a single linear design (effectively, a full mesh). However, adding one switch allows a much wider variety of candidates (e.g., 8x4 or 4x4x2), which could make the design feasible with fewer switch ports. Even if the specified number of switches is not prime, it might have inconvenient factors, that could be difficult to satisfy unless the number of ports per switch is quite large – e.g., $S = 94$ would require switches with at least $49+T$ ports, but $S = 95$ would work with $24 + T$-port switches.

We have not yet designed an algorithm to help optimize this parameter choice.

Algorithm 2 Optimal $\vec{K}$ assignment

```
1: Given:
2:   $D$: Number of dimensions of HyperX
3:   $\vec{S} = (S_1, S_2, ..., S_D)$: Size of each dimension of HyperX
4:   $R$: Switch radix,
5:   $T$: Number of terminals per switch,
6:
7: Initialize:
8:   $P = R - T$
9:   $\forall i \in [1, D], K_i = 0$
10:  found=TRUE
11:
12: while ($P > 0$) AND (found=TRUE) do
13:   minSK= $\min_{i=1}^{D} S_iK_i$
14:   found= FALSE
15:   for $i \in [1, D]$ do
16:     if ($S_iK_i$ = (minSK) AND ($P \geq S_i - 1$) then
17:       $K_i = K_i + 1$
18:       $P = P - (S_i - 1)$
19:       found=TRUE
20:
21: return $\vec{K} = (K_1, K_2, ..., K_D)$
```

6.2 Optimal HyperX $\vec{K}$ assignment

The bisection bandwidth of a HyperX$(D, \vec{S}, \vec{K}, T)$ depends both on the topology dimensions $\vec{S}$ and the per-dimension link bandwidth multipliers (LAG factors) $\vec{K}$. Here we show how to optimize $\vec{K}$. This is the same as finding an optimal distribution of each switch's available ports among the different dimensions, such that the bisection bandwidth is maximized.

Given: (i) switches with radix R, of which T ports are used for links to servers and (ii) a HyperX network with D dimensions, with sizes $\vec{S} = (S_1, S_2, ..., S_D)$. Our goal is to distribute the remaining $R - T$ ports of each switch among the D dimensions such that the bisection bandwidth of the topology is maximized. Note that for $HyperX(D, \vec{S}, \vec{K}, T)$, the bisection bandwidth is $\min_{i=1}^{D} S_iK_i$.

Problem: Maximize $\min_{i=1}^{D} S_iK_i$ under the following constraints:

$$\forall i, K_i \in \mathbb{Z} \tag{2}$$

$$\sum_{i=1}^{D}(S_i - 1)K_i \leq R - T \tag{3}$$

Our algorithm for assigning the K_is is shown in Algorithm 2. We first set $K_i = 0$ for all i, and initialize the number of spare ports P to $R - T$. At every step, we consider any dimension i with the minimal S_iK_i product. If enough spare ports are available to increase the bandwidth in that dimension, then we increment K_i by 1. Notice that we reduce the spare ports P by $S_i - 1$, as each switch connects to $S_i - 1$ switches in that dimension. We continue this until we do not have enough spare ports left to increase the bisection bandwidth.

We have a proof that this algorithm returns the optimal assignment of K_is, but it is too lengthy for this paper.

6.3 Generating Fat Tree topologies

As described in Section 3, we consider Extended Generalized Fat Trees (EGFTs) parametrized with number of levels L, a vector for the number of modules aggregated at each level $\vec{M}$, a vector for the number of top switches at each level $\vec{C}$, and a vector for the lag factor at each level $\vec{K}$. The goal of the algorithm presented here is to generate feasible EGFTs with a given number of switches, each with a fixed radix, and connect a given number of servers.

Construction constraints: Given switches with radix R, a EGFT$(L, \vec{M}, \vec{C}, \vec{K})$ can be constructed only if the following constraints hold:

- The top-most switches (level L top switches) should have enough ports to connect to all M_L $L-1$ EGFTs with K_L links.

$$M_L K_L \leq R \tag{4}$$

- At each level $1 \leq l < L$, a top switch should have enough ports to connect to all M_l $l-1$ EGFTs with K_l links, along with the ports to connect to the top switches at the $l+1$ level (we refer to these as "uplinks"). Note that there are C_{l+1} top switches at level $l+1$ with K_{l+1} downlinks, and the C_l top switches should have enough uplink ports to account for those downlinks.

$$1 \leq l < L, M_l K_l + \frac{C_{l+1} K_{l+1}}{C_l} \leq R \tag{5}$$

Finding suitable EGFTs: Given S switches with radix R and N servers, there are various EGFTs possible with different oversubscription ratios, maximum hop-counts, and numbers of cables required. We use a recursive procedure (pseudocode shown in Algorithm 3) to systematically explore the space of possible EGFTs. This $O(L^S)$ algorithm constructs EGFTs bottom-up, and currently it outputs all feasible EGFTs it finds. It is easy to modify this algorithm to generate only those EGFTs with oversubscription below a maximum threshold. For instance, setting this threshold to 1 outputs only the EGFTs with full bisection.

We start the recursion with the following inputs: an empty EGFT=$(0,,,)$, the number of modules NM=N, the number of uplinks at this lowest level NUP=1, and the number of remaining switches RS=S. If number of terminals per switch T is also provided, we run recursion with EGFT=$(1, \langle T\rangle, \langle 1\rangle, \langle 1\rangle)$, NM=$\lceil \frac{N}{T} \rceil$, NUP=$R-T$, and RS=$S-$NM.

In each recursive call, we add another level to the existing EGFT, until all modules are aggregated into one single topology (base case for recursion: NM== 1). Note that at each level $l+1$, this routine systematically explores all possible options for picking the number of lower level modules to aggregate M_{l+1}, the number of top switches to use C_{l+1}, and the bandwidth from each top switch to each module K_{l+1}. We make sure that the constraints in equations 4 and 5 are satisfied as we consider possible values for M_{l+1}, C_{l+1}, and K_{l+1}.

Algorithm 3 EGFTRecurse: Recursive function for constructing EGFTs

```
1: Input:
2:    EGFT: Current topology $(l, \vec{M}, \vec{C}, \vec{K})$
3:    NM: Number of modules at this level
4:    NUP: Number of uplinks available at each module
5:    RS: Remaining switches
6: Global:
7:    $R$: Switch radix
8:
9: /* Base case for recursion */
10: if NM $== 1$ then
11:    Output EGFT as a possible topology
12:
13: /* For each possible aggregation size */
14: for $2 \leq M_{l+1} \leq$MIN(NM, $R$) do
15:    /* For each possible number of top switches */
16:    for $1 \leq C_{l+1} \leq$ MIN(NUP, RS$/($NM$/M_{l+1})$) do
17:       /* For each possible $K$ value */
18:       for $1 \leq K_{l+1} \leq$ MIN($R/M_{l+1}$, NUP$/C_{l+1}$) do
19:          EGFTRecurse(
20:             EGFT($l+1, \vec{M} \cup M_{l+1}, \vec{C} \cup C_{l+1}, \vec{K} \cup K_{l+1}$),
21:             NM$/M_{l+1}$,
22:             $(R - (M_{l+1} * K_{l+1})) * C_{l+1}$,
23:             RS $-(C_{l+1}*$NM$/M_{l+1})$) ;
```

This recursive exploration can generate a lot of topologies. For example, an execution with $N = 1024, R = 48, T = 32$ results in more than 1 billion possible topologies. However, many of these topologies are clearly inferior, with respect to oversubscription ratios, to other similar topologies. Therefore, we implemented four heuristics to prune the output set:

- **H1**: If all modules are being aggregated (i.e., $M_{l+1} == NM$ in the pseudocode), then it does not make sense to consider all possible values for K_{l+1}. To minimize oversubscription, we need to consider only the maximum possible value for K_{l+1}.
- **H2**: Note that the oversubscription ratio of a topology is the maximum ratio across all levels. So, when building up a EGFT, we do not consider any assignment of M, C, and K that achieves a lower ratio than that has already been imposed by choices at the lower levels.
- **H3**: If all modules at a level can be aggregated into one module, i.e., switch radix R is greater than the number of modules NM at a level, then use the maximum aggregation size instead of trying smaller sizes. Smaller aggregation sizes increase the number of levels, consuming more switches and links without improving bisection bandwidth.
- **H4**: At the top-most level, we use the maximum possible and available top switches that use all uplinks at the next lower level, instead of iterating over all possible values for C.

Table 3: Reduction in search space with heuristics

Parameter			Heuristics used					
N	R	T	None	H1	H2	H3	H4	All
1K	48	16	>1.8B	303.9M	64.3M	315.6K	56.2M	521
1K	48	32	1.0B	61.8M	17.8M	16	13.3M	1
2K	48	16	>1.8B	1.2B	205.5M	471.4M	95.8M	31.0K
2K	48	32	>1.8B	220.1M	64.4M	87.9K	21.5M	521
1K	144	16	>1.9B	>1.5B	184.2M	128	>1.7B	1
1K	144	32	>1.9B	>1.5B	331.8M	112	>1.7B	1
2K	144	16	>1.9B	>1.5B	757.8M	128	>1.7B	1
2K	144	32	>1.8B	>1.5B	>994.5M	112	>1.7B	1
			Values above are sizes of the search space					

Table 3 shows how these heuristics, both independently and together, reduce the search space for several examples of N, R and T – by at least five orders of magnitude, for $N \leq 2048$. We explored up to 4 levels, and terminated incomplete jobs (shown as "$> x$") after 5 hours. Run-times when using all heuristics took less than 200msec, except for one case that took 6 sec.

7 Physical layout of data-center cables

In order to convert an abstract topology into a physical wiring plan, we must know the physical dimensions of the data center, including constraints on where cables can be run between racks.

We use Manhattan routing, rather than trying to route cables on diagonals, which could save some cable costs, but might be infeasible given typical cable plenums.

In some cases, the designer must choose between packing servers and switches as densely as possible into racks (generally a good idea, since POD or data-center floor space is often expensive), or ensuring that all server-to-switch cables stay within the server's rack (which can be useful if racks are shipped pre-wired.) We expose this policy choice, and its consequences (in terms of the number of racks required) to the designer.

We assume the use of standard-size racks (about 24 inches wide, 36 inches deep, and 78 inches tall). We assume that racks are arranged in rows; best practices for cooling call for no space between each rack in a row. Rows are separate either by "cold aisles" or "hot aisles" (i.e., either sources of cool air or sinks of heated air). Several considerations govern the choice of aisle widths [14], but generally the cold aisle must be at least 4 feet wide and the hot aisle at least 3 feet wide. For this paper, we assume that both aisles are 4 feet wide; extending our algorithm to handle mixed widths requires another set of decisions, and is future work.

In modern data centers, network cables do not run under raised floors, because it becomes too painful to trace underfloor cables when working on them. Therefore, inter-rack cables run in ceiling-hung trays above the racks. One tray runs directly above each row of racks, but there are relatively few trays running between rows, because too many cross trays are believed to excessively restrict air flow. We believe that the minimum reasonable separation between cross trays is about two rack widths. (We have not yet done the airflow simulations to validate this assumption.)

We note in passing that if the cables are cut much longer than necessary, the bundles of excess cable can create additional air-flow problems, and can also lead to space problems (not enough room), weight problems (especially for overhead trays), and installation problems (someone has to find a place to put these bundles, and to tie them down).

One other issue to note is that rack dimensions, and rack and tray spacings, are generally given in units of feet and inches, while standard cable lengths are in meters. We suspect this use of incommensurable units could lead to some complications in avoiding excess cable loops.

Algorithm 4 Algorithm for wiring cost computation

```
Given:
  $G_l(V_l, E_l)$: Logical topology
  $PMap(v)$: maps $v \in V_l$ to position $(x, y, z)$
  $RW$: Rack Width
  $CHTH$: Distance: top of the rack to ceiling-hung tray
  $G_{CT}$: Gap, in racks, between two cross trays
  $Cost(d)$: maps cable with length $d$ to its cost

Initialize:
  CableCost= 0

for $e = (v_1, v_2) \in E_l$ do
  len = 0
  $(x_1, y_1, z_1) \leftarrow PMap(v_1)$
  $(x_2, y_2, z_2) \leftarrow PMap(v_2)$

  /* Add length to pull the two ends of the cable to the side of the
  rack */
  len $+=$ $RW$

  if $x_1 == x_2$ AND $y_1 == y_2$ then
    /* both ends are in the same rack */
    len $+= |z_1 - z_2|$
  else
    /* not in the same rack; add length to reach trays */
    len $+= z_1 + z_2 + 2 * CHTH$

    if $x_1 \neq x_2$ AND $y_1 \bmod G_{CT} > 0$ AND $y_2 \bmod$
    $G_{CT} > 0$ AND $y_1/G_{CT} == y_2/G_{CT}$ then
      /* Exception: Manhattan distance does not work */
      distanceToCrossTray1 = $RW * (y_1 \bmod G_{CT} + y_2 \bmod$
      $G_{CT})$
      distanceToCrossTray2 = $RW * (2 * G_{CT} - (y_1 \bmod$
      $G_{CT} + y_2 \bmod G_{CT}))$
      len $+= |x_1 - x_2| +$ MIN(distanceToCrossTray1, dis-
      tanceToCrossTray2) ;
    else
      len $+= |x_1 - x_2| + |y_1 - y_2|$
  CableCost += $Cost$(len)

return CableCost
```

7.1 A general algorithm to create wiring plans

In this section, we describe an algorithm (Algorithm 4, in pseudo-code) to generate a physical wiring plans, including specific cable lengths, from an abstract logical topology. The algorithm also computes the total cost of the cables in the wiring plan, including the server-to-switch cables. The algorithm is generic; it works for all topology types.

The logical topology is provided as a graph $G_l(V_l, E_l)$, where the set of vertices V_l contains both servers and switches of the logical topology and edges E_l represents the links. Also given is a function $T(v)$ that provides the size of node $v \in V_l$ in terms of Rack Units (RU). One RU is 1.75 inches.

We assume that the designer provides the number of racks, their arrangement in rows (X rows with Y racks per row), and their two-dimensional spacing (in particular, the widths of the cold and hot aisles).

Therefore, the wiring problem is to figure out a feasible distribution of the servers and switches in the logical topology to the positions in the racks, such that the cable cost is the lowest. We denote the position of a node that is z RU from the top of the y-th rack in row x as (x, y, z).

Although Algorithm 4 is generic, it depends on a topology-specific function $PMap()$ that assigns servers and switches from the logical topology to a point in three-dimensional space. This is a difficult problem, and we discuss it in detail in Sec. 7.2.

The algorithm also depends on a generic function $Cost(len)$ which, for a cable of length len, determines the appropriate cable type and then computes a cable cost. We discuss this issue in Sec. 7.3.

Note that given two vertices v_1, v_2 and their positions $(x_1, y_1, z_1), (x_2, y_2, z_2)$ the Manhattan distance metric between these two positions is not enough to estimate the length of the cable needed in practice because of several reasons: (i) Switch ports and server NIC interfaces can be located anywhere in the middle of the rack, (ii) For proper airflow, cables are pulled to the side of the rack before they are routed up or down in a rack, (iii) Connections between racks have to run via ceiling-hung cable trays, (iv) Cable trays are a few feet – we assume 24 inches – above racks, (v) There are fewer cross trays (trays across the rows) than the number of racks in a row, to allow sufficient air flow.

To account for (i) and (ii), we add half of the rack width for each end of the link in cable length calculations. To account for (iii) and (iv), we consider the length for reaching the trays above the racks.

We note that even with constraint (v), except in one case, we can use the Manhattan distances between racks to compute the length of cables within the trays themselves. The only exception is when connecting two racks R_i and R_j in different rows, when neither rack is directly under a cross tray, but both R_i and R_j are between a pair of consecutive cross trays. In this case, we pick the cross tray that minimizes the cable length.

7.2 Logical to physical mapping functions

Algorithm 4 imports a topology-specific function $PMap(v)$ that assigns physical space positions to servers and switches in the logical topology ($v \in V_l$). Finding a $PMap(.)$ that minimizes cable cost is a hard problem (we believe it is NP-hard, but we have not yet worked out a reduction). The solution space is huge, because any permutation of nodes in the logical topology is a legal assignment for any subset of rack positions with size $|V_l|$.

We have designed heuristics for $PMap(.)$ for the HyperX and EGFT topology types.

For FatTree networks, we pack servers and switches in order to fill racks as much as possible. For HyperX networks, we chose instead to avoid any cable that runs between a server in one rack and a switch in another; this means that some of our HyperX results might require more racks than strictly necessary. In the future we will modify our algorithms to give the designer the choice between these options.

HyperX: Note that a HyperX topology is symmetric along its dimensions. Also, in HyperX topologies, all switches are edge switches. We leverage these observations to treat all the switches within a dimension as identical, which reduces the search space for rack space assignments. We consider all permutations (orderings) of the dimensions. Each permutation defines the sequence in which switches are assigned to racks. Suppose rh denotes the rack height and eh denotes the height of an edge switch plus the height of the associated servers. Then we can pack $\lfloor rh/eh \rfloor$ edge switches and associated servers into each rack, using the chosen sequences.

We currently assume $eh = 1 + T$ RU (i.e., servers and switches are all 1 RU high); this is probably wrong for high-radix switches, but not significantly wrong.

EGFT: We employ a simple heuristic for packing the nodes of an EGFT. We first pack the edge switches and associated servers in a fashion similar to the one we describe for HyperX, except that we do not have any dimensions in an EGFT on which to randomize. We then iterate from bottom to top and left to right of the logical topology, distributing the switches at each level to the first available space in each of the partially-populated racks. If no such rack is available, then we choose an empty rack. We fill rows before crossing between aisles; this heuristic seems to give shorter cable lengths.

7.3 Cable cost calculation

For the function $Cost(len)$ used in Algorithm 4, we must compute the cost of a cable of length len. We support either of two scenarios: standard-length cables, based on data such as in Table 2, or custom-length cables.

We first attempt to use a single-channel SFP copper cable for any logical link with LAG factor 1. For wider links, we use quad-channel QSFP copper cables for runs up to 5 meters, or quad-channel QSFP+ fiber cables for longer runs.

When using standard cables, we always choose a cable from the list of standard parts whose length is at least as long as required, and we obtain the cable cost from that list. Sometimes this results in a lot of excess cabling to hide, for longer cable runs.

When using custom cables, our current implementation calculates the cost for a cable of the exact length required.[7] We use the cost model for connectors and cables from Sec. 4.2. We then inflate these costs by an arbitrary 25% to account for the extra costs of purchasing custom cables and carrying them in inventory.

7.4 Choosing between copper and optical

Because high-speed optical network cables cost more than copper cables (see Table 2 for examples), normally we would prefer to use copper cables – if the cable length is below the limit on 10GbE copper cables, approximately 5 meters.

However, if plenum space is limited and we have to route a lot of cables, copper cables might not fit. In this case, we would need to replace some of the copper cables with optical cables.

We have deferred the solution of this problem to future work, especially because (based on some preliminary estimates) it does not appear to be a real problem for moderately large networks.

8 Examples of Perseus results

In this section, we present some results showing the ranges of cost vs. performance tradeoffs that Perseus exposes to the network designer. *(Remember, these results are based on only semi-plausible parts costs, and should never be quoted as realistic network costs.)*

For reasons of space, we limit the results presented here to configurations with N (servers) set to either 1024 or 8192. We consider a variety of switch radices: 32, 48, 64, 96, and 144, and we consider T (servers/switch) values of 16, 24, and 32. For HyperX networks, we explored designs with "excess" bisection bandwidth, since HyperX is not an "equal-cost multipath" topology, and optimal routing could be difficult.

Figures 1 and 2 show cost vs. bandwidth curves for HyperX and FatTree configurations for, respectively, 1K and 8K servers. Cost is based on our models for switch cost and for standard-length cables, including installation labor, and includes server-to-switch cables. For the HyperX curves, we plot curves for just a few values of S (number of switches) to preserve readability, and the points on each curve reflect various choices for D and $\vec{S}$. For the FatTree curves, the points on each curve reflect various choices for S, but we plot curves only for selected values of T, to keep the graphs readable. In all cases, we plot only the least-cost configuration that achieves a target bisection bandwidth. The curves show only the points for the least-cost physical layout for a given abstract topology. Note that naive designs could have much higher costs than the designs we generate.

Figures 1 and 2 lead to several observations. First, total network cost generally increases with increasing bandwidth, but not always; especially for larger HyperX networks, one can often find a "better" configuration at a much lower cost through a small parameter changes. Second, for a given target bandwidth, there are often several possible parameter choices with widely varying costs. Finally, although the figures suggest that FatTree networks might be somewhat less expensive than HyperX networks for the same target bandwidth, our cost models are currently too crude to support this as a general conclusion.

Total costs obviously depend on our models for parts costs; Fig. 3 shows how the HyperX($N = 8192$) results would change if switches cost just 20% as much as in Fig. 2 (the same low-cost model as in [13]). Note that the relative ranking of the curves usually does not change, but some of the "sweet spots" do.

Figure 4 shows how cost varies with HyperX network worst-case hop counts. (For FatTree networks, the worst-case hop count is simply $2L$; L=6 for all of the configurations plotted in Fig. 2.) One can sometimes, by spending more money, reduce the worst case hop count by one or two, but generally high-bandwidth topologies also have low hop counts.

Computation time: Computation costs are tolerable, especially since generating these graphs parallelizes easily. Table 4 shows elapsed times for various N.

Table 4: Computation costs (wall times)

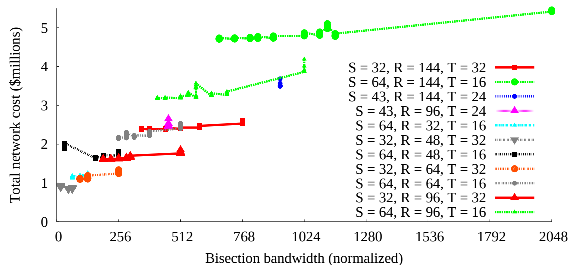
Network	N	Xeon CPUs	Jobs	Worst-case	Total
HyperX	8K	3GHz	166	155s	3.4hr
HyperX	16K	3GHz	68	20min	14.6hr
FatTree	8K	2.33GHz	96	55min	28.8hr

8.1 Secondary metrics

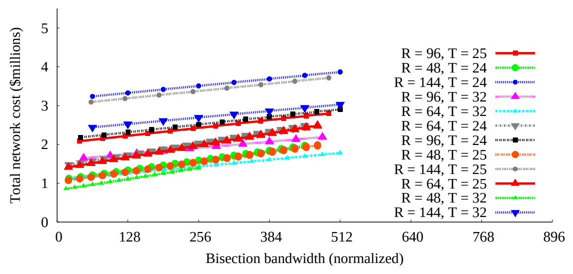
We found that using custom cables does not increase costs very much, based on our crude model for their parts cost. (Space does not permit us to show these graphs.)

However, custom cables do significantly reduce the amount of excess cable that must be tucked away without blocking airflow. For example, for HyperX

[7] We plan to modify this so that it quantizes the lengths in multiples of perhaps 0.5 meters, thus reducing SKU count at the cost of having to hide some slight excess cabling. This would also allow us to use standard cables if they are available in the right length.

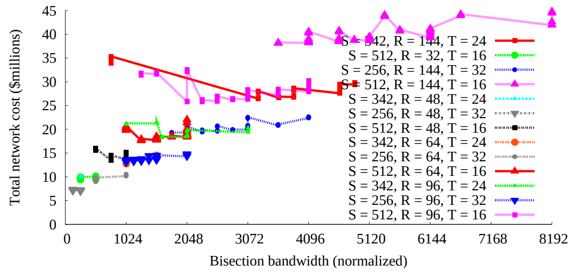


HyperX configurations

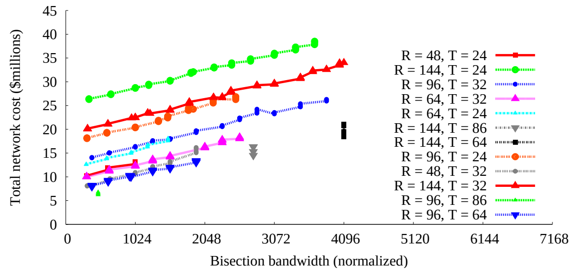


FatTree configurations

Figure 1: Cost vs. bisection bandwidth for 1024-server networks



HyperX configurations

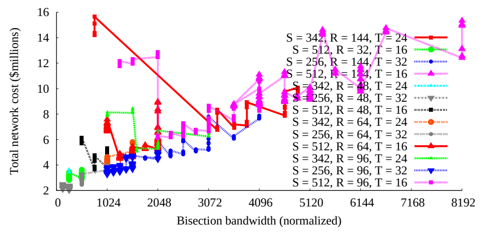


FatTree configurations

Figure 2: Cost vs. bisection bandwidth for 8192-server networks

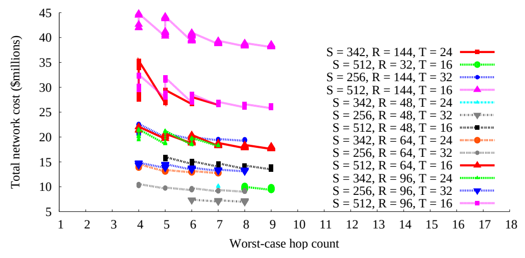
with N=8192, the least-cost full-bandwidth configuration with standard cables requires 9 SKUs, or 141 SKUs using custom cables. When using standard cables, HyperX/N=8192 configurations end up with mean per-cable excess lengths of between 23 and 265 in.

We also can quantify the number of wasted lanes in quad-channel cables. For HyperX with N=8192, this ranges from 0 to >55K, depending on the configuration.



HyperX configurations, $N = 8192$

Figure 3: Cost vs. BW for $100 switch ports



HyperX, $N = 8192$

Figure 4: Cost vs. worst-case hop count

9 Related work

Traditional topology analyses did not focus on physical layout issues that arise in a data center; they mostly considered only logical metrics such as bisection bandwidth. Much of the prior work also does not address the problem of efficiently sweeping through the logical topology space [6].

Ahn *et al.* [3] presented an algorithm that minimizes the number of switches needed to build a HyperX network with a given bisection. However, they do not consider any layout issues.

Popa *et al.* [13] compared the costs of several data-center network architecture families. Their analysis covered the use of server cores for switching, and included energy costs. However, they did not directly address the problem of finding a least-cost network design within a specific topology family, and they do not appear to have tried to optimize the placement of switches in racks.

Farrington *et al.* [8] analyzed cabling issues for FatTree networks. They showed that much cost could be eliminated by consolidating the upper levels of a FatTree, replacing cables, connectors, and a physically distributed set of low-radix commodity switches with a design using merchant silicon. Many of the cables become traces on circuit boards. They also show how to use higher-speed links within a FatTree to reduce the cable count.

The Helios [8] design also addresses the costs of cabling, switching, and especially the costs of electrical to optical conversions. Helios focuses on the connections between containers, and exploits low-cost, relatively slow optical switches, so it covers a somewhat different domain than Perseus is aimed at.

Currently our tools do not model topology related energy costs. At one level, quantifying power is easy: we simply add up the power consumption of the individual switches. However, as switch designers improve the energy-proportionality of their products, switch power becomes more dependent on traffic demands (that is the goal of proportionality). An accurate estimate of network-related power consumption requires a detailed, time-varying traffic model, and also requires accurate models for switch power consumption at various traffic levels. These data are often hard to obtain, which could make it difficult to compare topology-related energy costs; Popa *et al.* [13] used average network-wide utilization as a proxy. Abts *et al.* [1] demonstrated that a flattened butterfly topology is more power-efficient than a folded-Clos (FatTree) network.

Prior work by Heller *et al.* [10] has shown that dynamic adjustment of the set of active, power-consuming links can increase network-wide proportionality. Thus, there is a complex interaction between network topology, traffic demands, and power consumption, and we do not know how to model this issue in detail.

10 Future work

Within our framework, there is a lot of room for further work, including:

- Understanding how to model internal vs. external bandwidth. Currently, we assume that local servers and external connections properly share the overall bisection bandwidth of a network, but this is probably wrong. We also need to understand whether designers care where the external ports can be connected.
- Dealing different widths for hot and cold aisles.
- Incorporating plenum capacity into copper vs. fiber choice and/or design-feasibility testing (See Sec. 7.4).
- Modeling network energy consumption. This requires models for traffic and switch energy proportionality.
- Modeling network reliability.

11 Summary and conclusions

In this paper we have exposed the complexity of the problem of choosing good designs for high-bandwidth, multi-path data-center networks. We have described the Perseus framework for assisting network designers in exploring this space, and we have presented several algorithms that help to optimize parameter choices. We have also shown that, based on semi-plausible parts costs, the overall cost for constructing a network with a given bisection bandwidth can vary significantly.

We know from past experience that the relative costs of parts such as switches and cables will change over time, sometimes dramatically. We also expect NIC and switch port speeds to increase in several jumps. These trends mean that topology choices based on current parts will undoubtedly need to be re-evaluated every year or two; thus, our goal in this paper has been to provide a methodology and a set of algorithms, not to describe the "correct" choices for topologies and their parameters.

While designers of real networks will undoubtedly use different costs, they will still have to grapple with the choice of parameters, and Perseus should prove useful in this task.

References

[1] D. Abts, M. R. Marty, P. M. Wells, P. Klausler, and H. Liu. Energy Proportional Datacenter Networks. In *ISCA*, pages 338–347, 2010.

[2] H. Abu-Libdeh, P. Costa, A. Rowstron, G. O'Shea, and A. Donnelly. Symbiotic Routing in Future Data Centers. In *SIGCOMM*, 2010.

[3] J. Ahn, N. Binkert, A. Davis, M. McLaren, and R. S. Schreiber. HyperX: Topology, Routing, and Packaging of Efficient Large-Scale Networks. In *Proc. Supercomputing*, Nov. 2009.

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

[5] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat. Hedera: Dynamic Flow Scheduling for Data Center Networks. In *Proc. NSDI*, Apr. 2010.

[6] W. J. Dally and B. Towles. *Principles and Practices of Interconnection Networks*. Morgan Kaufman, 2004.

[7] N. Farrington, G. Porter, S. Radhakrishnan, H. H. Bazzaz, V. Subramanya, Y. Fainman, G. Papen, and A. Vahdat. Helios: A Hybrid Electrical/Optical Switch Architecture for Modular Data Centers. In *SIGCOMM*, 2010.

[8] N. Farrington, E. Rubow, and A. Vahdat. Data Center Switch Architecture in the Age of Merchant Silicon. In *Proc. Hot Interconnects*, pages 93–102, 2009.

[9] 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 *Proc. SIGCOMM*, Barcelona, 2009.

[10] B. Heller, S. Seetharaman, P. Mahadevan, Y. Yiakoumis, P. Sharma, S. Banerjee, and N. McKeown. ElasticTree: Saving energy in data center networks. In *Proc. NSDI*, 2010.

[11] R. N. Mysore, A. Pamboris, N. Farrington, N. Huang, P. Miri, S. Radhakrishnan, V. Subramanya, and A. Vahdat. PortLand: A Scalable Fault-Tolerant Layer 2 Data Center Network Fabric. In *Proc. SIGCOMM*, 2009.

[12] S. Öhring, M. Ibel, S. Das, and M. Kumar. On generalized fat trees. In *Proc. Parallel Proc. Symp.*, 1995.

[13] L. Popa, S. Ratnasamy, G. Iannaccone, A. Krishnamurthy, and I. Stoica. A Cost Comparison of Data Center Network Architectures. In *Proc. CoNEXT*, Dec. 2010.

[14] N. Rasmussen and W. Torell. Data Center Projects: Establishing a Floor Plan. White Paper 144, American Power Conversion, 2007.