Topology Switching for Data Center Networks

Kevin C. Webb, Alex C. Snoeren, and Kenneth Yocum
UC San Diego

Abstract

Emerging data-center network designs seek to provide physical topologies with high bandwidth, large bisection capacities, and many alternative data paths. Yet, existing protocols present a one-size-fits-all approach for forwarding packets. Traditionally, the routing process chooses one "best" route for each end-point pair. While some modern protocols support multiple paths through techniques like ECMP, each path continues to be selected using the same optimization metric. However, today's data centers host applications with a diverse universe of networking needs; a single-minded forwarding approach is likely to either let paths go unused, sacrificing reliability and performance, or make the entire network available to all applications, sacrificing needs such as isolation.

This paper introduces *topology switching* to return control to individual applications for deciding best how to route data among their nodes. Topology switching formalizes the simultaneous use of multiple routing mechanisms in a data center, allowing applications to define multiple *routing systems* and deploy individualized *routing tasks* at small time scales. We introduce the topology switching abstraction and illustrate how it can provide both network efficiency and individual application performance, and admit flexible network management strategies.

1 Introduction

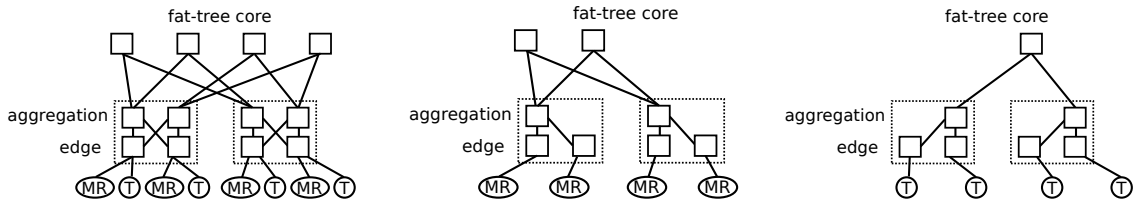
Data center networking architectures are rapidly evolving to accommodate the demands of the "cloud computing" model, where cloud providers dynamically rent raw computing resources or application services to clients. In particular, data center networks support an increasingly sophisticated environment that includes storage, monitoring, data processing, and virtual machine management software. These services place different performance, reliability, and management demands on the underlying network. For example, while caching services (e.g., memcached [6]) or HPC workloads prize low latency communication, data processing applications (e.g., MapReduce [4]) require high bisection bandwidth.

Unfortunately, application performance and network management suffer from the one-size-fits-all routing design prevalent in today's data center networks. A common approach to managing data center networks is to segregate traffic into separate VLANs, based on application or organizational unit. However, inside a VLAN each application suffers the same fate: a single or *unified* routing process chooses the "best" route for each end-point pair irrespective of the applications running in the VLAN. Thus the fate of all VLAN communication, no matter how different, is bound to the properties of one routing system. For example, traditional switched Ethernet domains provide silos of transparent connectivity, routing across a single spanning tree, resulting in non-shortest paths, idle alternative paths, and high link stress at the tree root.

This paper makes the case for *topology switching* (TS), a fundamentally different way for data center applications to interact with the network. A topology-switched network supports multiple, simultaneous application-specific routing tasks, somewhat like VLANs. Unlike VLANs, however, within each routing task, the application can define distinct topologies, naming, and routing conventions specifically tailored to their unique reliability, performance, and scalability requirements, providing enhanced performance at the expense of certain routing attributes that may not be critical to the routing task at hand. Thus, the data center network no longer needs to balance routing consistency, failure resilience, high performance forwarding, flexible policy enforcement, and security across all the applications in the data center.

Topology switching attempts to address the challenges raised by two key data center network management processes: VM placement and network evolution. While virtualization allows any server to host any VM, the network is often a key performance bottleneck [11]. The majority of proposed data center networks try to decouple placement from performance by maximizing bandwidth between server pairs through symmetric topologies. While this strategy admits considerable flexibility in assigning work across the data center, there are cases in which skewed communication patterns dominate. We argue that for these workloads it is far more effective to select application-specific routing systems to make efficient use of physical resources.

Individual VM pools can host a variety of services that each present different networking demands. For example, the hosted services themselves may be clustered,



(a) The physical data center network. (b) A topology for a MapReduce task. (c) A topology for a trading platform.

Figure 1: Optimizing topologies for individual applications, topology switching finds a fat-tree subgraph for MapReduce (MR) and an isolated spanning tree for the trading platform (T).

multi-tiered, or replicated. Each of these design patterns performs best on a distinct routing system—a well-connected, low-diameter mesh, a high-bandwidth tree of arbitrary depth, and an edge-disjoint, redundant multigraph, respectively.

Additionally, real-world data center infrastructures grow and evolve, often becoming less homogeneous and symmetric. This heterogeneity may result from growth, limited budgets, physical wiring constraints, or the presence of dynamic link allocation via optics [5] or wireless [14]. As the topology distorts, routing designs based upon a systematic network design may begin to perform poorly—or not at all. Topology switching functions over any physical topology, allowing applications to optimize for the network at hand. Finally, the needs of data center applications are changing quickly. For example, Amazon recently introduced explicit support for high-performance computing (HPC, applications that often have latency-bound phases) instances on EC2, presumably in response to sufficiently strong economic incentives.

This paper lays the groundwork for the design and development of a topology-switched network architecture. We use a simulation framework to show that topology switching allows routing systems to optimize for different metrics, even for a randomized placement of hosts across the physical infrastructure. Many implementation challenges remain, however, including managing the costs and complexity of dynamic reconfiguration in real switching hardware, as well as designing a management interface that provides a straightforward way for operators to configure and debug topology switched networks.

2 Topology switching

Topology switching allows applications to create custom network topologies to meet their specific requirements. Consider the fat-tree data center network in Figure 1(a) connecting eight physical hosts. Various emerging multi-path routing systems support well-provisioned networks [1, 3, 13, 20], but their unified routing systems remain blind to individual application needs. Such networks may host a range of applications inside distinct VM pools, such as a bandwidth-hungry MapReduce/Hadoop

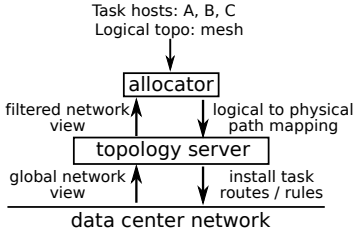


Figure 2: A topology server mediates access to the network, compiling individual tasks and performing admission control.

cluster (MR) and a queuing-sensitive trading platform (T). While the MapReduce application is generally bottlenecked by the available bisection bandwidth in the network, trading platforms demand the consistent low-latency performance of isolated network paths [2].

In contrast, a topology-switched network treats these two applications as distinct routing *tasks*. Each routing task runs an instance of a particular routing *system* that best addresses the communication pattern and preferences of that particular task. A routing system includes an *allocator* that determines the subset of the physical network that will connect the application endpoints in its task. In this case, the MapReduce task in Figure 1(b) searches for high-bandwidth physical paths between mappers and reducers to optimize the performance of the shuffle phase. In contrast, the trading platform in Figure 1(c) allocates for isolation, building an exclusively-owned spanning tree. Routing systems also define a set of route selection rules that allow switches to make application-specific forwarding decisions across multiple paths.

Figure 2 illustrates the process of compiling routing tasks in a topology-switched network. Routing tasks specify the set of communicating end hosts within the data center, a desired logical topology to construct between those hosts, and a routing system to manage link allocation and route selection. A logically centralized (but possibly replicated) topology server registers each system, compiles routing tasks, and manages individual task deployment. Topology allocators take as input the node set, a desired logical topology, and a view of physical network connectivity from the topology server. The allocator then maps one or more physical paths to each link in the logical

topology to achieve its performance objectives. Allocation occurs in an on-line fashion, and individual allocators are not allowed to change existing allocations.

The topology server influences allocation in many ways. First, the server may prune links or switches from the physical network view before passing it to the allocator, allowing network administrators to export different views to each routing task. This mechanism makes it trivial to physically separate traffic from other tasks. For example, the topology server can remove the spanning tree links in Figure 1(c) from the network view passed to the MapReduce allocator. The server may also perform admission control on the task, refusing to deploy the routing task or revoking instantiated tasks. This flexibility additionally gives administrators a mechanism by which to upgrade the physical network between task allocations.

3 Allocation strategies

Topology switching allows network operators to customize tasks along three primary axes: logical topology, allocator, and route selection rules. Here, we set aside route selection and employ simple, hash-based multi-path route selection for all tasks. Additionally, we only consider logical mesh networks. Other logical topologies, such as rings (for chain replication) or trees (for aggregation or file distribution), could leverage allocation to customize the arrangement of the topology, e.g., to build an efficient aggregation tree. Within these constraints, we present three allocators that optimize for resilience, isolation, and bandwidth, respectively.

Allocators have three components. First, each allocator uses one or more metrics to define an objective function that drives the mapping process. The second component is an allocation algorithm to maximize or minimize the objective function. The third component annotates and optionally filters the network substrate. Since allocation is an on-line process, an allocation's goodness may decrease as other allocations (optimizing for different metrics) are made. Thus the allocator may wish to store additional annotations on the substrate views passed to future allocations. The topology server may optionally filter links based on these annotations, ensuring that other allocators do not decrease the mapping quality for prior allocations (for an example see Section 3.3).

Formally, the physical network is a graph $P = \{V, E\}$, where V is the set of hosts and switches and E is the set of physical links. A logical topology $T = \{H, L\}$ is a set of hosts $H \subseteq V$ and logical links L connecting hosts in H. Given a view of the substrate, P^{view}, allocators map each logical link l_i to a set of paths p_i in that view. The topology server maintains a set of annotations for each physical link $e_i \in E$, including the total number of tasks mapped to this link, its physical capacity (C_i), and a number of *claims* (M_i) made to that capacity (Section 3.1).

3.1 Bandwidth

Capacity is often a core objective of unified routing architectures. These topologies are well suited for parallel data processing tasks that optimize for low all-to-all transfer times and high bisection bandwidth. Other bandwidth-oriented tasks include data backup, video and audio serving, and VM image distribution.

Metric: Two metrics used to evaluate data center network designs are bisection bandwidth and all-to-all transfer time. The bisection bandwidth of a topology is the amount of bandwidth a bijection of hosts can transfer to one another; it is a rough measure of a topology's ability to handle many concurrent transfers. All-to-all transfer time, in contrast, measures the time taken for each node to transfer an x MB file to every other node. It reflects a worst-case communication scenario and the effects of applied load.

Allocation: An allocation strategy may consider either single or multiple path solutions, where multiple physical paths support the logical link. In either case, one allocation strategy is to maximize the total flow possible along links (end-to-end paths) in the logical topology. However, even a single-path solution must consider how other logical links in this task have been mapped, otherwise many logical links could be mapped onto the same physical links. This approach can be modeled as a maximum multi-commodity flow problem, where polynomial time solutions exist when allowing fractional flow allocations.

However, for simplicity we approximate this allocation by using a single-path allocator. This allocator uses a maximum spanning tree to find the current maximal path [16]. The allocator depends upon an estimate of available bandwidth on each link, which depends on the number of *claims* to the link's capacity.

Substrate annotation/filtering: The topology server manages claim annotations, M_i, and uses them to set the available capacity on links as $\frac{C_i}{M_i}$. In this work, bandwidth tasks increment M_i by one for each physical link mapped to a logical link. In contrast, resilience and isolation allocators increment M_i by $1/N$ when a logical link is backed by N physical paths, dividing a bandwidth share equally across them. This allows the bandwidth allocator to take advantage of the capacity left by allocations that are not bandwidth constrained. This does assume that the network has the ability to rate-limit endpoints, perhaps using emerging VM-based technologies [9, 19].

3.2 r resilience

An allocator may also wish to increase the overall physical path diversity available to its logical topology. For example, consider aggregation trees used for scalable monitoring [15]. These tasks are willing to traverse longer paths in return for increased failure resilience, ensuring more hosts are connected during failures or congestion.

The large numbers of components in modern data center networks means that some level of failure is virtually always present [7]. This is perhaps even more of a concern with emerging topologies that depend on end hosts to participate in switching [8, 10].

Metric: Here we measure resilience as the number of cuts r in the physical substrate required to break a logical link. Since hosts have a single up-link to their top-of-rack switch in the physical topologies we study, we ignore cuts to those access links for pairs of hosts on different switches. We note that this approach provides an aggressive notion of resilience, providing r disjoint paths (not including access links) between nodes.

Allocation: Here we use shortest paths to find a suitable set of r paths between endpoints. For each logical link we repeatedly find a shortest path with respect to hop count, add it to the set of possible paths, and remove its links from consideration (a residual graph on P^{view}). The task may also specify an average resilience that allocation must reach to succeed, allowing some logical paths to be backed by fewer than r disjoint physical paths.

Substrate annotation/filtering: Beyond setting the link claims as described above, this allocator does not require additional annotations. Future allocations will not decrease the resilience of allocated tasks.

3.3 k isolation

Isolation between tasks may be used to ensure consistent levels of network performance, or to isolate critical services from potentially disruptive external traffic sources. However, modern data center switches are often limited in their ability to provide per flow isolation. While there may be hundreds to thousands of separate routing tasks, current switches support only a handful of fair queuing classes (8 according to [19]). Applications may use the isolation allocator to ensure sufficient resources exist to provide isolation.

Metric: We measure isolation on each physical link as the number of routing tasks with a logical path on that link. Here each task specifies the maximum number of other tasks that may share a link, k. k may either reflect the maximum number of service classes the physical network can support or it may be set to one to provide complete isolation from other traffic.[1]

Allocation: To increase isolation, we wish to find a topology that connects the task's nodes that is minimally shared with other tasks. We approximate the aggregate level of sharing by summing the total number of tasks allocated across the chosen physical links. This task is equivalent to finding a minimum-cost tree connecting the hosts, a Steiner tree on an existing graph, an NP-hard problem. We employ the minimum spanning tree heuristic, with a worst-case performance ratio of 2.

[1] A similar scheme could provide isolation on a per-switch basis.

The allocator computes the minimum spanning tree on the substrate, using the number of current tasks as the edge weight. The allocator then removes all physical links and switches from this spanning tree that are not used in the paths between nodes in the task. Each link used in the mapped topology increases its task count.

Substrate annotation/filtering: Unfortunately, subsequent allocations may violate the isolation requirement. To prevent this, the topology server can remove links from consideration. In this case, the topology server first removes links whose number of tasks is already k. To do so, the topology server records the minimum k used by an isolation task for each link. $k = inf$ for links with no isolation tasks. Note that removing links may disconnect P^{view} and subsequent allocations may fail.

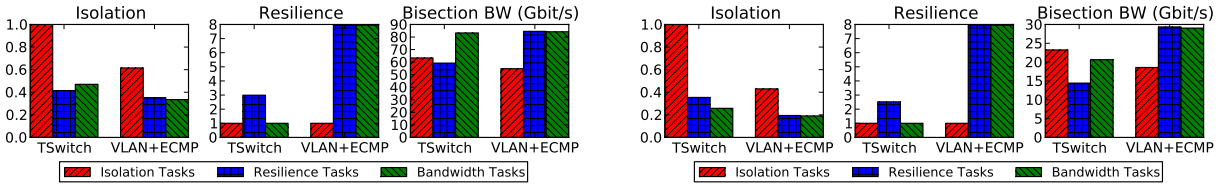
4 Simulations

A fundamental question posed by topology switching is whether allocating paths for specific objectives can improve the networks for all tasks relative to a unified routing system. To begin to answer, we built a simulator that takes as input a physical network graph, P, and multiple routing tasks. We simulate the three allocators described in Section 3 and report how well each task met its objective (and those of the other allocators).

Obviously, the results depend on the particular physical topology and set of routing tasks. Hence, the resulting metrics are only meaningful in comparison to an alternative design. While traditional Ethernet's single spanning tree is an obvious choice, it is an exceptionally weak strawman as it fails to take advantage of the path diversity and resulting bisection bandwidth made available by recently proposed physical topologies.

Instead, we compare our topology switched allocations to a unified routing system modeled after recent proposals such as Trill [20], 802.1aq [13], and Cisco's FabricPath product [3]. These systems route layer-2 frames using multiple shortest paths calculated by instances of the IS-IS routing protocol. We emulate this ECMP approach by calculating all shortest paths in substrate P for each pair of end hosts. We then assume that the pair can achieve the maximum flow along the union of those paths. This outperforms standard hash-based ECMP, which is oblivious to available capacity and limits the number of considered paths (16 for FabricPath).

We evaluate allocations based on metrics for bisection bandwidth, isolation, and resilience. We calculate isolation using the number of logical paths each task assigned to each physical link. For a given routing task, we calculate isolation as the ratio of the task's paths to the total path count on the link, averaged across all physical links. A value of 1 indicates a highly isolated allocation while near zero values indicate that other tasks dominate the link. We measure resilience as described in Section 3.2.



(a) Performance of a 2i-2r-2b task mix.

(b) Performance of a 7i-5r-4b task mix.

Figure 3: Topology switching versus ECMP emulation on a $k = 16$ fat tree with 1024 hosts.

To judge the bandwidth quality of the mapped logical topology, we consider each task's *effective* bisection bandwidth. Since task allocation often results in asymmetric topologies, we calculate bisection bandwidth as the average maximum flow between two randomly chosen sets of $|H|/2$ nodes (repeated 200 times). Note that maximum flow depends upon the capacity of the physical links in the mapped logical topology. To take into account the existence of the other tasks, we weight the physical capacity by the total claims (Section 3.1) made on this link by this task divided by all claims made on the link.

4.1 Experiments

We study topology switching on a $k = 16$ fat tree with 16-port switches and 16 pods. All links are 1 Gb/s in capacity. We create a pool of routing tasks, each allocated using one of our three routing systems. Each task uses an exclusive set of end hosts randomly distributed across the physical network. We use this "network oblivious" host-to-task assignment to determine if topology switching can meet different objectives without optimizing physical layout. All tasks have equal node count and use a mesh logical network. Finally, our experiments allocate isolation before resilience and bandwidth tasks, ensuring that those tasks find $k = 1$ isolations.[2] Note that ECMP emulation uses the isolation allocator to identify a VLAN, as if an administrator planned the task. However, unlike the isolation allocator, ECMP emulation allows successive tasks to use links from previously allocated "VLANs."

We first investigate allocations for six tasks: two isolation, two resilience, and two bandwidth (2i-2r-2b). Figure 3(a) shows the average results for each metric for the isolation, resilience, and bandwidth tasks respectively. Looking at topology switching ("TSwitch"), all tasks achieve their goals. First, isolation tasks achieve their target ($k = 1$). In contrast, "VLAN+ECMP" provides less isolation for all tasks, though our allocator increases isolation relative to the other tasks. Interestingly, topology switching provides increased isolation for the resilience and bandwidth tasks as well. This is likely because we remove isolation task links from consideration and our bandwidth allocator uses fewer paths than ECMP.

Second, in topology switching, resilience tasks receive exactly $r = 3$ disjoint paths through the topology. ECMP on the other hand treats resilience and bandwidth tasks identically, spreading logical paths across as many shortest paths as exist, giving both high resilience. Finally, the topology-switched bandwidth tasks receive the highest effective bisection bandwidth, illustrating that the other allocators are giving up bandwidth relative to their optimization goals.

It is also important to qualify the kind of bandwidth ECMP emulation reports. In a fat tree, this emulation gives near optimal effective bisection bandwidth by using every minimum hop-count path, and the average number of multiple paths for ECMP emulation is over 60 (std. dev. of 13.5) per logical path for both 2i-2r-2b and 7i-5r-4b task mixes. Second, the topology-switched isolation tasks receive more bandwidth than their counterpart tasks under ECMP emulation. More importantly, that bandwidth *is not shared* with any other tasks mapped to the topology. In other words, the "trading platform" from Section 2 gets high performance by virtue of its isolation.

The results are similar, Figure 3(b), when allocating seven isolation, five resilience, and four bandwidth tasks (7i-5r-4b). However, by using seven isolation tasks, seven of the 8 uplinks of the fat tree's edge switch cannot be allocated to other tasks when using topology switching. This reduces the ability of topology switching to deliver high effective bisection bandwidth to the bandwidth task, a 31% decrease relative to the ECMP emulation.[3] However, the topology-switched isolation tasks continue to receive superior capacity; a trade-off a unified routing system cannot achieve. This could be further improved with a multi-commodity flow bandwidth allocator, rather than our current approximation scheme.

Finally, topology switching and ECMP utilize the network similarly for the 2i-2r-2b mix. However, topology switching the 7i-5r-4b task mix allows 10% of the links to go unused in the fat tree. This is essentially the penalty of supporting many isolation requests.

[2]This emulates a topology server that runs isolation tasks on a network view that only shows isolation allocations. It would then reallocate other tasks that conflict with the new mapping.

[3]Note that by hosting 16 tasks of 64 nodes, the maximum possible bisection bandwidth is 32 Gb/s.

5 Related work

Several recently proposed systems find paths from a global view of the network, but they use a bandwidth-centric heuristic for all routes. For instance, SPAIN's [17] path discovery is closely related to our r resilience allocator. Hedera [1] estimates flow demand to assign flows to paths in multi-rooted tree topologies to increase throughput for large flows. Similar throughput-maximizing allocations dynamically provision the physical network topology in systems like Helios [5] (optics) and Flyway [14] (wireless). Part of our future work is exploring the substrate filtering polices network operators could use to provision dynamic resources between routing tasks.

One project that presents a unique allocator (that is not maximally bandwidth driven) is ElasticTree [12]. ElasticTree allocates a "minimal-power" network subset that attempts to satisfy the current traffic matrix while minimizing the use of power-hungry physical network components. The authors observe that load varies over time for particular applications in their data center traces, allowing them to identify network components to shut down. In contrast, a topology-switched network would find other routing tasks that could benefit from the free resources.

The allocation phase of topology switching is closely related to the virtual network embedding problem found in many virtualized network models [18, 21]. In those systems, users provide precise network requirements: specific links, capacities, delay, and router processing resources. Such systems either guarantee a network slice with those requirements or reject the request; the focus is to support a modest number of arbitrary network protocols, providing each an isolated network that persists for extended periods. In contrast, a topology-switched network multiplexes tens to hundreds of application-specific topologies that are requested in terms of relative network characteristics and may have short lifetimes (minutes).

6 Conclusion

As Section 4 discusses, a range of routing technologies now exist to route across the multiple paths found in well-connected topologies. While they increase performance by spreading flows for all applications in a similar fashion, such unified treatment makes it difficult or impossible to deliver qualitatively different (fewer, longer but fatter, or less-well-provisioned) paths to applications.

Topology switching rejects the one-size-fits-all approach to path selection to provide applications with the ability to have a stake in their routing decisions. We describe one possible set of allocators that allow applications to take advantage of paths that others choose not to utilize. Further, our initial results show that we can leverage almost the full diversity of a fat tree topology to meet application objectives.

References

[1] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat. Hedera: Dynamic flow scheduling for data center networks. In *NSDI*, 2010.

[2] A. Bach. Higher- and faster-than-ever trading to transform data center networks. Datacenter Dynamics, 2009.

[3] Cisco. *Scaling Data Centers with FabricPath and the Cisco FabricPath Switching System*, 2010.

[4] J. Dean and S. Ghemawat. MapReduce: Simplified data processing on large clusters. In *OSDI*, 2004.

[5] 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.

[6] B. Fitzpatrick. Distributed caching with memcached. *Linux Journal*, (124), 2004.

[7] A. Greenberg, J. R. Hamilton, et al. VL2: A scalable and flexible data center network. In *SIGCOMM*, 2009.

[8] 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. *SIGCOMM*, 2009.

[9] C. Guo, G. Lu, H. Wang, S. Yang, C. Kong, P. Sun, and W. n. Z. Y. Wu. Secondnet: A data center network virtualization architecture with bandwidth guarantees. In *ACM CoNEXT*, 2010.

[10] 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.

[11] J. Hamilton. Data center networks are in my way. Talk: Stanford Clean Slate CTO Summit, 2009.

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

[13] IEEE. 802.1aq - shortest path bridging, 2010. http://www.ieee802.org/1/pages/802.1aq.html.

[14] S. Kandula, J. Padhye, and P. Bahl. Flyways to de-congest data center networks. In *HotNets*, 2009.

[15] D. Logothetis and K. Yocum. Wide-scale data stream management. In *USENIX*, 2008.

[16] N. Malpani and J. Chen. A note on practical construction of maximum bandwidth paths. *Inf. Process. Lett.*, 83:175–180, August 2002.

[17] J. Mudigonda, P. Yalagandula, M. Al-fares, and J. C. Mogul. Spain: Cots data-center ethernet for multipathing over arbitrary topologies. In *NSDI*, 2010.

[18] R. Ricci, C. Alfeld, and J. Lepreau. A solver for the network testbed mapping problem. *ACM SIGCOMM CCR*, 32(2), 2003.

[19] A. Shieh, S. Kandula, A. Greenberg, and C. Kim. Seawall: performance isolation for cloud datacenter networks. In *HotCloud*, 2010.

[20] J. Touch and R. Perlman. RFC 5556: Transparent interconnection of lots of links (TRILL), May 2009.

[21] M. Yu, Y. Yi, J. Rexford, and M. Chiang. Rethinking virtual network embedding: Substrate support for path splitting and migration. *ACM SIGCOMM CCR*, 38(2), April 2008.