SPAIN: Design and Algorithms for Constructing Large Data-Center Ethernets from Commodity Switches

Jayaram Mudigonda, Praveen Yalagandula, Mohammad Al-Fares, Jeffrey C. Mogul

HP Laboratories
HPL- 2009-241

Keyword(s):
Ethernet, Scalable, COTS, multipath

Abstract:
Operators of data centers want a scalable network fabric that supports high bisection bandwidth and host mobility, but which costs very little to purchase and administer. Ethernet almost solves the problem – it is cheap and supports high link bandwidths – but traditional Ethernet does not scale, because its spanning-tree topology forces traffic onto a single tree. Many researchers have described "scalable Ethernet" designs to solve the scaling problem, by enabling the use of multiple paths through the network. However, most such designs require specific wiring topologies, which can create deployment problems, or changes to the network switches, which could obviate the commodity pricing of these parts.

In this paper, we describe SPAIN ("Smart Path Assignment In Networks"). SPAIN provides multipath forwarding using inexpensive, commodity off-the-shelf (COTS) switches, over arbitrary topologies. SPAIN pre-computes a set of paths that exploit the redundancy in a given network topology, then merges these paths into a set of trees; each tree is mapped as a separate VLAN onto the physical Ethernet. SPAIN requires only minor end-host software modifications, including a simple algorithm that chooses between pre-installed paths to efficiently spread load over the network. We demonstrate SPAIN's ability to improve bisection bandwidth over both simulated and real data-center networks.

External Posting Date: October 8, 2009 [Fulltext]
Internal Posting Date: October 8, 2009 [Fulltext]

Approved for External Publication

SPAIN: Design and Algorithms for Constructing Large Data-Center Ethernets from Commodity Switches

Jayaram Mudigonda* Jayaram.Mudigonda@hp.com
Praveen Yalagandula* Praveen.Yalagandula@hp.com
Mohammad Al-Fares+ malfares@cs.ucsd.edu
Jeffrey C. Mogul* Jeff.Mogul@hp.com

*HP Labs, Palo Alto, CA 94304 +UC San Diego

Abstract

Operators of data centers want a scalable network fabric that supports high bisection bandwidth and host mobility, but which costs very little to purchase and administer. Ethernet *almost* solves the problem – it is cheap and supports high link bandwidths – but traditional Ethernet does not scale, because its spanning-tree topology forces traffic onto a single tree. Many researchers have described "scalable Ethernet" designs to solve the scaling problem, by enabling the use of multiple paths through the network. However, most such designs require specific wiring topologies, which can create deployment problems, or changes to the network switches, which could obviate the commodity pricing of these parts.

In this paper, we describe SPAIN ("Smart Path Assignment In Networks"). SPAIN provides multipath forwarding using inexpensive, commodity off-the-shelf (COTS) switches, over arbitrary topologies. SPAIN precomputes a set of paths that exploit the redundancy in a given network topology, then merges these paths into a set of trees; each tree is mapped as a separate VLAN onto the physical Ethernet. SPAIN requires only minor end-host software modifications, including a simple algorithm that chooses between pre-installed paths to efficiently spread load over the network. We demonstrate SPAIN's ability to improve bisection bandwidth over both simulated and real data-center networks.

1 Introduction

Data-center operators often take advantage of scale, both to amortize fixed costs, such as facilities and staff, over many servers, and to allow high-bandwidth, low-latency communications among arbitrarily large sets of machines. They thus desire scalable data-center networks. Data-center operators also must reduce costs for both equipment and operations; commodity off-the-shelf (COTS) components often provide the best total cost of ownership (TCO).

Ethernet is becoming the primary network technology for data centers, especially as protocols such as Fibre Channel over Ethernet (FCoE) begin to allow convergence of all data-center networking onto a single fabric. COTS Ethernet has many nice features, especially ubiquity, self-configuration, and high link bandwidth at low cost, but traditional Layer-2 (L2) Ethernet cannot scale to large data centers. Adding IP (Layer-3) routers "solves" the scaling problem via the use of subnets, but introduces new problems, especially the difficulty of supporting dynamic mobility of virtual machines. The lack of a single flat address space makes it much harder to move a VM between subnets. Also, the use of IP routers instead of Ethernet switches can increase hardware costs and complicate network management.

This is not a new problem; plenty of recent research papers have proposed scalable data-center network designs based on Ethernet hardware. All such proposals address the core scalability problem with traditional Ethernet, which is that to support self-configuration of switches, it forces all traffic into a single spanning tree [29] – even if the physical wired topology provides multiple paths that could, in principle, avoid unnecessary sharing of links between flows.

In Sec. 3, we discuss previous proposals in specific detail. Here, at the risk of overgeneralizing, we assert that SPAIN improves over previous work by providing multipath forwarding using inexpensive, COTS Ethernet switches, over arbitrary topologies, and supporting incremental deployment; we are not aware of previous work that does all four.

Support for COTS switches probably reduces costs, and certainly reduces the time before SPAIN could be deployed, compared to designs that require even small changes to switches. Support for arbitrary topologies is especially important because it allows SPAIN to be used without re-designing the entire physical network, in contrast to designs that require hypercubes, fat-trees, etc., and because there may be no single topology that best meets all needs. Together, both properties allow incremental deployment of SPAIN in an existing data-center network, without reducing its benefits in a purpose-built network. SPAIN can also function without a real-time central controller, although it may be useful to exploit

such a controller to achieve certain QoS goals.

In SPAIN, a network management system first precomputes a set of paths that exploit the redundancy in a given network topology, and then merges these paths into a set of trees, with the goal of utilizing the redundancy in the physical wiring both to provide high bisection bandwidth (low over-subscription), and to support several failover paths between any given pair of hosts. Each tree is then mapped onto a separate VLAN, exploiting the VLAN support in COTS Ethernet switches. In most cases, only a small number of VLANs suffices to cover the physical network.

SPAIN does require modifications to end-host systems, including a simple algorithm that chooses between pre-installed paths to efficiently spread load over the network. These modifications are quite simple; we describe several alternatives, including an in-kernel version using an existing Linux module, and a user-mode version that requires no kernel changes.

We have evaluated SPAIN both in simulation and in an actual deployment on a network testbed. We show that SPAIN adds only minor end-host overheads, that it supports near-optimal bisection bandwidth on a variety of topologies, that it can be deployed incrementally with immediate performance benefits, and that it tolerates faults in the network.

2 Background and goals

Ethernet is known for its ease-of-use. Hosts come with preset addresses and simply need to be plugged in; each network switch automatically learns the locations of other switches and of end hosts. Switches organize themselves into a spanning tree to form loop-free paths between all source-destination pairs. Hence, not surprisingly, Ethernet now forms the basis of virtually all enterprise and data center networks. This popularity made many Ethernet switches an inexpensive commodity and led to continuous improvements. 10Gbps Ethernet is fast becoming commoditized [19], the first 40Gbps commercial version is expected by the end of 2009 [16], and the standardization of 100Gbps is already underway [6].

Network operators would like to be able to scale Ethernet to an entire data center, but it is very difficult to do so, as we detail in section 2.2. Hence, today most such networks are designed as several modest-sized Ethernets (IP subnets), connected by one or two layers of IP routers [3, 4, 10].

2.1 Why we want Ethernet to scale

The use of multiple IP subnets, especially within a data center, requires one to carefully fuse together, and then manage the interactions of, a variety of network architectures based on very different forwarding, control and administrative approaches. Consider a typical network composed of Ethernet-based IP subnets. This not only requires the configuration of IP subnets and routing protocols—which is considered a hard problem in itself [22]—but also sacrifices the simplicity of Ethernet's plug-and-play operation. For instance, as explained in [3, 4], in such a hybrid network, to allow the end hosts to efficiently reach the IP-routing layer, all Ethernet switches must be configured such that their automatic forwarding table computation is forced to pick only the shortest paths between the IP-routing layer and the hosts.

Dividing a data center into a set of IP subnets has other drawbacks. It imposes the need to configure DHCP servers for each subnet; to design an IP addressing assignment that does not severely fragment the IP address space (especially with IPv4); and makes it hard to deal with topology changes [22]. For example, migrating a live virtual machine from one side of the data center to another, to deal with a cooling imbalance, requires changing that VM's IP address – this can disrupt existing connections.

For these reasons, it becomes very attractive to scale a single Ethernet to connect an entire data center or enterprise.

2.2 Why Ethernet is hard to scale

Ethernet's lack of scalability stems from three main problems:

1. Its use of the Spanning Tree Protocol to automatically ensure a loop-free topology.
2. Packet floods for learning host locations.
3. Host-generated broadcasts, especially for ARP.

We discuss each of these issues.

Spanning tree: Spanning Tree Protocol (STP) [29] was a critical part of the initial success of Ethernet; it allows automatic self-configuration of a set of relatively simple switches. Using STP, all the switches in an L2 domain agree on a subset of links between them, so as to form a spanning tree over all switches. By forwarding packets only on those links, the switches ensure connectivity while eliminating packet-forwarding loops. Otherwise, Ethernet would have had to carry a hop-count or TTL field, which would have created compatibility and implementation challenges.

STP, however, creates significant problems for scalable data-center networks:

- **Limited bisection bandwidth**: Since there is (by definition) only one path through the spanning tree between any pair of hosts, a source-dest pair cannot use multiple paths to achieve the best possible bandwidth. Also, since links on any path are probably shared by many other host pairs, congestion can arise, especially near the root of the tree. The aggregate throughput of the network can be much lower than the sum of the NIC throughputs.
- **High-cost core switches**: Aggregate throughput

can be improved by use of a high-fanout, high-bandwidth switch at the root of the tree. Scaling root-switch bandwidth can be prohibitively expensive [10], especially since this must be replicated to avoid a single point of failure for the entire data center. Also, the STP must be properly configured to ensure that the spanning tree actually gets rooted at this expensive switch.

- **Low reliability**: Since the spanning tree leads to lots of sharing at links closer to the root, a failure can affect an unnecessarily large fraction of paths.
- **Reduced flexibility in node placement**: Generally, for a given source-dest pair, the higher the common ancestor in the spanning tree, the higher the number of competing source-dest pairs that share links in the subtree, and thus the lower the throughput that this given pair can achieve. Hence, to ensure adequate throughput, frequently-communicating source-dest pairs must be connected to the same switch, or to neighboring switches with the lowest possible common ancestor. Such restrictions, particularly in case of massive-scale applications that require high server-to-server bandwidth, inhibit flexibility in workload placement or cause substantial performance penalties [10, 19].

SPAIN avoids these problems by employing multiple spanning trees, which can fully exploit the path redundancy in the physical topology, especially if the wiring topology is not a simple tree.

Packet floods: Ethernet's automatic self-configuration is often a virtue: a host can be plugged into a port anywhere in the network, and the switches discover its location by observing the packets it transmits [35]. A switch learns the location of a MAC address by recording, in its learning table, the switch port on which it first sees a packet sent from that address. To support host mobility, switches periodically forget these bindings and re-learn them.

If a host has not sent packets in the recent past, therefore, switches will not know its location. When forwarding a packet whose destination address is not in its learning table, a switch must "flood" the packet on all of its ports in the spanning tree (except on the port the packet arrived on). This flooding traffic can be a serious limit to scalability [2, 22].

In SPAIN, we use a mechanism called *chirping* (see Sec. 6) which avoids most timeout-related flooding.

Host broadcasts: Ethernet's original shared-bus design made broadcasting easy; not surprisingly, protocols such as the Address Resolution Protocol (ARP) and the Dynamic Host Configuration Protocol (DHCP) were designed to exploit broadcasts. Since broadcasts consume resources throughout a layer-2 domain, broadcasting can limit the scalability of Ethernet broadcasting [4, 15, 22, 26]. Greenberg *et al.* [17] observe that "...the overhead of broadcast traffic (e.g., ARP) limits the size of an IP subnet to *a few hundred servers*..."

SPAIN does not eliminate broadcasts, but we can exploit certain aspects of both the data-center environment and our willingness to modify end-host implementations. See Sec. 11 for more discussion.

3 Related Work

Spanning trees in Ethernet have a long history. The original algorithm was first proposed in 1985 [29], and was adapted as the IEEE 802.1D standard in 1990. Since then it has been improved and adapted along several dimensions. While the Rapid Spanning Tree Protocol (802.1s) reduces convergence time, the Per-VLAN Spanning Tree (802.1Q) improves link utilization by allowing each VLAN to have its own spanning tree. Sharma *et al.* exploit these multiple spanning trees to achieve improved fault recovery [36]. In their work, Viking manager, a central entity, communicates and pro-actively manages both switches and end hosts. Based on its global view of the network, the manager selects (and, if needed, dynamically re-configures) the spanning trees.

Most proposals for improving Ethernet scalability focus on eliminating the restrictions to a single spanning tree.

SmartBridge, proposed by Rodeheffer *et al.* [32], completely eliminated the use of a spanning tree. SmartBridges learn, based on the principles of diffused computation, locations of switches as well as hosts to forward packets along the shortest paths. STAR, a subsequent architecture by Lui *et al.* [24] achieves similar benefits while also facilitating interoperability with the 802.1D standard. Perlman's RBridges, based on an IS-IS routing protocol, allow shortest paths, can interoperate with existing bridges, and can also be optimized for IP [30]. Currently, this work is being standardized by the TRILL working group of IETF [7]. Note that TRILL does not support multiple paths, because IS-IS provides only shortest-path routes.

Myers *et al.* [26] propose to eliminate the basic reason for the spanning tree, the reliance on broadcast as the basic primitive, by combining link-state routing with a directory for host information.

More recently, Kim *et al.* [22] propose the SEATTLE architecture for very large and dynamic Ethernets. Their switches combine link-state routing with a DHT to achieve broadcast elimination and shortest-path forwarding, without suffering the large space requirements of some of the prior approaches; otherwise, SEATTLE is quite similar to TRILL.

Greenberg *et al.* [19] propose an architecture that scales to 100,000 or more servers. They exploit programmable commodity layer-2 switches, allowing them

to modify the data and control planes to support hot-spot-free multipath routing. A sender host for each flow consults a central directory and determines a random intermediary switch; it then bounces the flow via this intermediary. When all switches know of efficient paths to all other switches, going via a random intermediary is expected to achieve good load spreading.

Several researchers have proposed specific regular topologies that support scalability. Fat trees, in particular, have received significant attention. Al-Fares *et al.* [10] advocate combining fat trees with a specific IP addressing assignment, thereby supporting novel switch algorithms that provide high bisection bandwidth without expensive core switches. Mysore *et al.* [27] update this approach in their PortLand design, which uses MAC-address re-writing instead of IP addressing, thus creating a flat L2 network. Scott *et al.* [34] similarly use MAC-address re-writing in MOOSE, but without imposing a specific topology; however, MOOSE uses shortest-path forwarding, rather than multipath.

VL2 [18] provides the illusion of a large L2 network on top of an IP network with a Clos [14] topology, using a logically centralized directory service. VL2 depends on both equal-cost multipath (ECMP) forwarding and IP Anycast addressing in a manner that is only applicable to Clos topologies; we do not believe VL2's approach applies to most other topologies.

The commercial switch vendor Woven Systems [8] also used a fat tree for the interconnect inside their switch chassis, combing their proprietary vScale chips with Ethernet switch chips that include specific support for fat-trees [5]. The vScale chips use a proprietary algorithm to spread load across the fat-tree paths.

In contrast to fat-tree topologies, others have proposed recursive topologies such as hypercubes. These include DCell [21] and BCube [20].

SPAIN differs from all of this prior work because it uses unmodified COTS switches, works with arbitrary topologies, supports incremental deployment, and requires no centralized controllers.

4 The design of SPAIN

We start with our specific goals for SPAIN, including the context in which it operates. Our goals are to:

- Deliver more bandwidth and better reliability than spanning tree.
- Support arbitrary topologies, not just fat-tree or hypercube, and extract the best bisection bandwidth from any topology.
- Utilize unmodified, off-the-shelf, commodity-priced (COTS) Ethernet switches.
- Minimize end host software changes, and be incrementally deployable.

In particular, we want to support flat Layer-2 addressing and routing, so as to:

- Simplify network manageability by retaining the plug-and-play properties of Ethernet at larger scales.
- Facilitate non-routable protocols, such as FibreChannel over Ethernet (FCoE) that are required for "fabric convergence" within data centers [23]. Fabric convergence, the replacement of special-purpose interconnects such as FibreChannel with standard Ethernet, can reduce hardware costs, management costs, and rack space.
- Improve the flexibility of virtual server and storage placement within data centers, by reducing the chances that arbitrary placement could create bandwidth problems, and by avoiding the complexity of VM migration between IP subnets.

We explicitly limit the focus of SPAIN to data-center networks, rather than trying to solve the general problem of how to scale Ethernet. Also, while we believe that SPAIN will scale to relatively large networks, our goal is not to scale to arbitrary sizes, but to support typical-sized data-centers.

4.1 Overview of SPAIN

In SPAIN, we pre-compute a set of paths that exploit the redundancy in a given network topology, and merge these paths into a set of trees, with the goal of utilizing the redundancy in the physical wiring both to provide high bisection bandwidth and to improve fault tolerance. We map each tree to a separate VLAN, and install these VLANs on the switches. We usually need only a few VLANs to cover the physical network, since a single VLAN tree can include multiple disjoint subtrees.

SPAIN allows a pair of end hosts to use different VLANs, potentially traversing different links at different times, for different flows; hence, SPAIN can achieve higher throughput and better fault-tolerance than traditional spanning-tree Ethernet.

SPAIN reserves VLAN 1 to include all nodes. This default VLAN is thus always available as a fallback path, or if we need to broadcast or multicast to all nodes. We believe that we can support multicast more efficiently by mapping multicast trees onto special VLANs, but this is future work.

SPAIN requires only a few features that are already present in most COTS switches: MAC-address learning and VLAN support. Optionally, SPAIN can exploit other switch features to improve performance, scale, and fault tolerance, or to reduce manual configuration: LLDP; SNMP queries to get LLDP information; and Per-VLAN Spanning Tree or Multiple Spanning Tree (see Sec. 5.6). For data centers where MAC addresses are known a priori, we have designed another approach

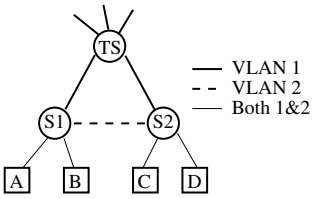


Figure 1: Example of VLANs used for multipathing

called *FIB-pinning*, which is described in more detail in Section 5.7.

Fig. 1 illustrates SPAIN with a toy example, which could be a fragment of a larger data-center network. Although there is a link between switches S1 and S2, the standard STP does not forward traffic via that link. SPAIN creates two VLANs, with VLAN1 covering the normal spanning tree, and VLAN2 covering the alternate link. Once the VLANs have been configured, end-host A could (for example) use VLAN1 for flows to C while end-host B uses VLAN2 for flows to D, thereby doubling the available bandwidth versus traditional Ethernet. (SPAIN allows more complex end-host algorithms, to support fault tolerance and load balancing.)

Note that TRILL or SEATTLE, both of are shortest-path protocols, would only use the path corresponding to VLAN2.

SPAIN requires answers to three questions:

1. Given an arbitrary topology of links and switches, with finite switch resources, how should we compute the possible paths to use between host pairs?
2. How can we set up the switches to carry these paths?
3. How do pairs of end hosts choose which of several possible paths to use?

Thus, SPAIN includes three key components, for *path computation*, *path setup*, and *path selection*. The first two can run offline (although online reconfiguration could help improve network-wide QoS and failure resilience); the path selection process runs online at the end hosts for each flow.

5 Offline configuration of the network

In this section, we describe the centralized algorithms SPAIN uses for offline network configuration: path computation and path setup. (Sec. 6 discusses the online, end-host-based path selection algorithms.)

These algorithms address several challenges:

- **Which set of paths to use?**: The goal is to compute smallest set of paths that exploit all of the redundancy in the network.
- **How to map paths to VLANs**: We must minimize the number of VLANs used, since Ethernet only allows 4096 VLANs, and some switches support fewer. Also, each VLAN consumes switch resources – a switch needs to cache a learning-table entry for each known MAC on each VLAN.
- **How to handle unplanned topology changes**: Physical topologies (links and switches) change either due to failures and repairs of links and switches, or due to planned upgrades. Our approach is to recompute and re-install paths only during upgrades, which should be infrequent, and depend on dynamic fault-tolerance techniques to handle unplanned changes.

We discuss these issues in detail.

5.1 Practical issues

SPAIN's centralized configuration mechanism must address two practical issues: learning the actual topology, and configuring the individual switches with the correct VLANs.

Switches use the Link-Layer Discovery Protocol (LLDP) (IEEE Standard 802.1AB) to advertise their identities and capabilities. They collect the information they receive from their neighbors and store it in their SNMP MIB. We can leverage this support to programmatically determine the topology of the entire L2 network.

Switches maintain a *VLAN-map* table, to track the VLANs allowed on each physical interface, along with information about whether packets will arrive with a VLAN header or not. Each interface can be set in *untagged mode* or *tagged mode* for each VLAN.[1] If a port is in *tagged* mode for a VLAN v, packets received on that interface with VLAN tag v in the Ethernet header are accepted for forwarding. If a port is in *untagged* mode for VLAN v, all packets received on that port without a VLAN tag are assumed to be part of VLAN v. Any packet with VLAN v received on a port not configured for VLAN v are simply dropped. For SPAIN, we assume that this VLAN assignment can be performed programmatically using SNMP.

For each graph computed by the path layout program, SPAIN's switch configuration module instantiates a VLAN corresponding to that graph onto the switches covered by that VLAN. For a graph $G(V, E)$ with VLAN number v, this module contacts the switch corresponding to each vertex in V and sets all ports of that switch whose corresponding edges appear in E in tagged mode for VLAN v. Also, all ports facing end-hosts are set to tagged mode for VLAN v, so that tagged packets from end-hosts are accepted.

5.2 Path-set computation

Our first goal is to compute a *path set*: a set of link-by-link loop-free paths connecting pairs of end hosts through the topology.

[1] This is the terminology used by ProCurve. CISCO uses *access* and *trunk* mode.

Algorithm 1 Algorithm for Path Computation

```
 1: Given:
 2:   G_full = (V_full, E_full): The full topology,
 3:   w: Edge weights,
 4:   s: Source, d: Destination
 5:   k: Desired number of paths per s, d pair
 6:
 7: G(V, E) = Compact(G_full);  /* graph compaction */
 8:
 9: Initialize: ∀e ∈ E : w(e) = 0
10: /* shortest computes weighted shortest path */
11: Path p = shortest(G, s, d, w) ;
12: for e ∈ p do
13:   w(e)+ = |E|
14: end for
15:
16: while (|P| < k) do
17:   p = shortest(G, s, d, w)
18:   if p ∈ P then
19:     /* no more useful paths */
20:     break ;
21:   end if
22:   P = P ⋃ {p}
23:   for e ∈ p do
24:     w(e)+ = |E|
25:   end for
26: end while
27:
28: return P
```

A good path set achieves two simultaneous objectives. First, it exploits the available topological redundancy. That is, the path set includes enough paths to ensure that any source-destination pair, at any given time, can find at least one *usable* path between them. By "usable path", we mean a path that does not go through bottlenecked or failed links. Hence a path set that includes *all* possible paths is trivially the best, in terms of exploiting the redundancy. However, such a path set might be impractical, because switch resources (especially on COTS switches) might be insufficient to instantiate so many paths. Thus, the second objective for a good path set is that it has as few paths as possible.

We accomplish this in three steps, as shown in Algorithm 1. First (Line 7), we compact the topology graph to reduce the number of nodes (and hence edges), which speeds up the subsequent steps. Otherwise, a network with tens of thousands of nodes might involve billions of source-destination pairs [11, 33].

Computing paths for each of these pairs is unnecessary and wasteful. We can derive an essential subset of this graph, much smaller than the original, based on two observations:

- First, we need to compute paths only between edge switches. This is because an end-to-end host-to-host path can be trivially determined by extending, at either end, the corresponding edge-switch-to-edge-switch path with the two access links.
- Second, wherever possible we remove *chains* of nodes. By "chain" we mean a series of nodes that form a line topology in two dimensions; each node, except for the those at the ends, connects to exactly two other nodes (on right and left sides). We can remove chains because they do not contribute to path diversity, even though they increase the number of source-destination pairs.

Given that the complexity of computing the shortest paths across all pairs is $O(|V|^3)$, where V is the set of nodes, these two optimizations yield huge computation savings. As quantified in Section 8, these two simple optimizations reduce the node count by a factor of 2.85 on average, and as much as 8.6 on one of the topologies on which we simulated these algorithms.

Second (Lines 9–14), we initialize the set of paths for each source-destination pair to include the shortest path. Shortest paths are attractive because, they minimize the network resources needed for each packet, and in general, have a higher probability of staying usable after failures. That is, under the simplifying assumption that each link independently fails (either bottlenecks or goes down) with a constant probability f, then a path p of length $|p|$ will be usable with probability $P_u(p) = (1 - (1 - f)^{|p|})$. (We informally refer to this probability, that a path will be usable, as its "usability," and similarly for the probability of a set of paths between a source-destination pair.) Clearly, since the shortest path has the smallest length, it will have the highest P_u.

Finally (Lines, 16–22), we grow the path set to meet the desired degree (k) of path diversity between any pair of hosts. Note that a path set is usable if at least one of the paths is usable. We denote the usability of a path set *ps* as $PS_u(\text{ps})$. This probability depends not only on the lengths of the paths in the set, but also on the degree of shared links between the paths. A best path set of size k has the maximum $PS_u(\cdot)$ of all possible path sets of size k. However, it is computationally infeasible to find the best path set of size k. Hence, we use a greedy algorithm that adds one path at a time, and that prefers the path that has the minimum number of links in common with paths that are already in the path set.

We prefer adding a link-disjoint path, because a single link failure can not simultaneously take down both the new path and the existing paths. However, the longer the new path, the smaller the additional improvement in the path set's usability, since $(1 - (1 - f)^{|p|})$ decreases with increasing $|p|$. Thus, in some circumstances it may be beneficial to prefer a path with shared links but which is much shorter than a fully link-disjoint path. Our algorithm, however, always prefers link-disjoint paths because, as shown below in Sec. 5.2.1, in most networks

with realistic topologies and operating conditions, a link-disjoint path improves the usability of a path set by the largest amount.

As shown in lines 12–14 and 23–25, we implement our preference for link-disjoint paths by incrementing the edge weights of the path we have added to the path set by a large number (number of edges). This ensures that the subsequent shortest-path computation picks a link that is already part of the path set only if it absolutely has to.

5.2.1 Benefits of link-disjoint paths

Algorithm 1 gives preference to link-disjoint paths, over alternative paths that are shorter but which have links in common with one or more already-selected paths. Here we explain why.

Consider a path set with a single path A of length a. Let paths X and Y, with lengths x and y respectively, be the next two equivalent candidate paths. The paths X and Y are equivalent in that the two sets $\{A, X\}$ and $\{A, Y\}$ have the same probability of being usable, given that any link fails with a probability f. Suppose X shares exactly one link with A, whereas Y shares none (i.e., Y is fully link-disjoint with A). Let $u = (1 - f)$, the per-link probability that the link will be usable. The probability that both paths A and X in the path-set $\{A, X\}$ will be down simultaneously is

$$PS_d(\{A, X\}) = (1-u)+u(1-u^{a-1})(1-u^{x-1}) \quad (1)$$

Since paths A and Y do not share any common links, the probability of both being down at the same time is

$$PS_d(\{A, Y\}) = (1 - u^a)(1 - u^y) \quad (2)$$

Since paths X and Y have same probability of usability, this implies

$$PS_d(\{A, X\}) = P_d(\{A, Y\}) \quad (3)$$

Then y can be derived as $y = [ln(u^{(x-1)}-c)/ln(u)]+1$, where c is the constant $[u^{(a-1)}(1 - u)/(1 - u^a)]$.

Figure 2 plots the length of an equivalent link-disjoint path (y in the derivation above) for a given length of a path that shares a single link (x in the derivation above). We fix a at 5, approximately the median path length for all of the topologies that we simulated (see Sec. 8), and consider two somewhat conservative estimates for $p_u = 1 - PS_d(\{A, X\})$ (or equivalently, $p_u = 1 - PS_d(\{A, Y\})$), 0.95 and 0.99.

It can be seen that for a link-disjoint path to become undesirable, it must be substantially longer than the corresponding shared-link path. For instance, any link-disjoint path that is less than 10 hops long is better than a shared-link path of length 5; the link-disjoint path can be 27 hops long if p_u is 0.99. This shows that even a very long link-disjoint path is preferable over a shorter but link-sharing path.

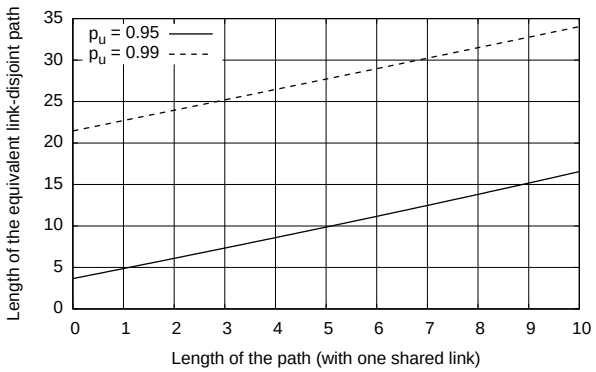


Figure 2: *Comparison of candidate paths.*

5.3 Mapping path sets to VLANs

Given a set of paths with the desired diversity, SPAIN must then map them onto a minimal set of VLANs. (Remember that there are at most 4096 possible VLANs on an Ethernet, and some switches might not have the resources to support even that many.)

We need to ensure that the subgraphs formed by the paths of each VLAN are loop-free, so that the switches work correctly in the face of forwarding-table lookup misses. On such a lookup miss for a packet on a VLAN v, a switch will flood the packet to all outgoing interfaces of VLAN v – if the VLAN has a loop, the packet will circulate forever. (We could run the spanning-tree protocol on each VLAN to ensure there are no loops, but then there would be no point in adding links to the SPAIN VLANs that the STP would simply remove from service.)

Notation: Given a graph $G(V, E)$, a path p is defined as an ordered set of vertices $(p^1, p^2, ..., p^{|p|})$ where $p^i \in V$ for $1 \leq i \leq |p|$, $(p^j, p^{j+1}) \in E$ for $1 \leq j < |p|$, and $|p|$ denotes the length of the path p.

Problem 1. VLAN Minimization: *Given a set of paths $P = \{p_1, p_2, ..., p_n\}$ in a graph $G = (V, E)$, find an assignment of paths to VLANs, with minimal number of VLANs, such that the subgraph formed by the paths of each VLAN is loop-free.*

5.3.1 VLAN Minimization is NP-Hard

In this section, we show that the VLAN Minimization problem is NP-hard. For the proof, we consider a more restricted version of this problem where all paths in P end at the same destination node. We refer to such path sets as *unique-destination path sets*. We show that this restricted problem is NP-hard. Thus it follows that the generalized VLAN Minimization is NP-hard.

Formally, a unique-destination path set $P = \{p_1, p_2, ..., p_n\}$ satisfies the following property: $p_i^{|p_i|} =$

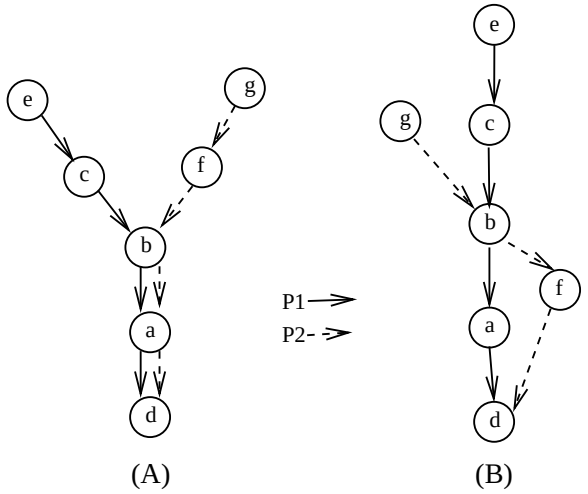


Figure 3: Example illustrating vlan-compatibility. Paths P1 and P2 are vlan-compatible in case (A) and not in case (B).

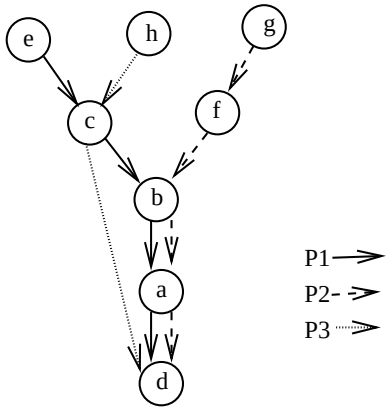


Figure 4: Example illustrating the non-transitivity of vlan-compatibility operator.

$p_j^{|p_j|}$ for $1 \leq i, j \leq n$. In such a path set, we define two paths to be *vlan-compatible* if and only if any common node on those two paths have the same next hop. Formally, we define *vlan-compatibility* for two paths as follows:

Definition 1. *Given two paths, p_1 and p_2, in a unique-destination path set, we define p_1 and p_2 to be* vlan-compatible *if and only if for all $1 \leq i < |p_1|$ and $1 \leq j < |p_2|$, $(p_1^i = p_2^j) \Rightarrow (p_1^{i+1} = p_2^{j+1})$.*

In plain words, two paths to a destination are *vlan-compatible* if once they join, they never again go separate ways until reaching the destination. We demonstrate this with different examples in Figure 3.

The above definition can be easily extended to a set of paths.

Definition 2. *A unique-destination path set $P = \{p_1, p_2, ..., p_k\}$ is defined to be* vlan-compatible *if and only if, for all $1 \leq i \leq k$ and $1 \leq j \leq k$, p_i and p_j are* vlan-compatible.

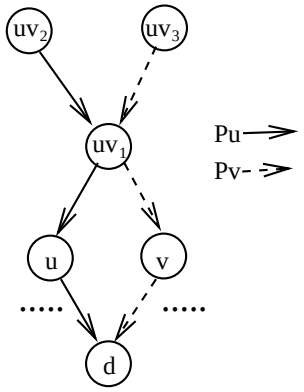


Figure 5: Example illustrating the construction step of a VLAN Minimization instance for a link (u, v) in a Vertex Coloring instance.

Note that *vlan-compatibility* is not a transitive relation. See Figure 4 for a contradictory example where $\{p_1, p_2\}$ and $\{p_2, p_3\}$ are *vlan-compatible* but p_1 and p_3 are not. Hence we need to check the *vlan-compatibility* between every pair in a path set before we can declare that path set to be *vlan-compatible*.

Problem 2. Restricted VLAN Minimization: *Given a unique-destination path set $P = \{p_1, p_2, ..., p_n\}$ in a graph $G = (V, E)$, find an assignment of VLANs to each of these paths such that a minimal number of VLANs are used and the subgraph formed by the paths in each VLAN is loop-free.*

Theorem 1. *Problem 2 is NP-hard.*

Proof Outline. We prove this by polynomial reduction from the Graph Vertex-Coloring problem, which is known to be NP-complete [39]. In a Vertex-Coloring decision problem, given an undirected graph $G = (V, E)$ and k colors, decide if each vertex $u \in V$ can be colored with one of the k colors such that no adjacent vertex v of u, i.e., $(v, u) \in E$, is assigned the same color as u. We reduce this decision problem to an instance of the Restricted VLAN Minimization Problem 2.

We construct a graph $G' = (V', E')$ and a unique-destination path set P from a given vertex coloring problem $G = (V, E)$. We construct P such that all paths end in a destination vertex d. We will construct one path in P for each vertex in the graph coloring instance. Our goal is to build G' and P such that two paths p_1 and p_2 in P are vlan-compatible if and only if the corresponding vertices in the graph coloring instances do *not* have a link between them.

Initially $V' = d \bigcup V$ and $E' = \bigcup_{v \in V}\{(v, d)\}$. For each vertex $u \in V$, we initialize a path p_u with just two nodes: (u, d) and add it to P. Then, for each link $(u, v) \in E$, we add three nodes uv_1, uv_2, uv_3 to G' and extend p_u and p_v as follows: $p_u = (uv_2, uv_1) \cdot p_u$ and $p_v = (uv_3, uv_1) \cdot p_v$, where $(\cdot)$ denotes the ordered-concatenation operator. We also add the relevant four links to E'. Figure 5 illustrates this step. Note that before this step, path p_u and p_v have a unique first node. Hence, this construction ensures that path p_u and path p_v are not *vlan-compatible*. At the end of this construction for all edges, any two nodes $a, b \in V$ s.t. $(a, b) \notin E$, p_a and p_b will be *vlan-compatible*.

A solution to this VLAN Minimization instance will give the minimum number of VLANs m needed for setting up those paths. If $m \leq k$, then we return answer `yes` to the vertex coloring problem, otherwise we return `no`.

By construction, $|V'| = 1 + |V| + 3|E|$ and $|E'| = |V| + 4|E|$. Hence, the resulting VLAN Minimization instance is polynomial in terms the space and in terms of the construction time. Hence, we have shown polynomial reducibility from graph vertex coloring to the restricted VLAN Minimization problem. □

Theorem 2. *The VLAN Minimization problem 1 is NP-hard.*

Proof. This problem is a generalized version of Problem 2, which we prove is NP-hard in Theorem 1. Hence, VLAN Minimization problem is NP-hard. □

5.3.2 A heuristic greedy algorithm

Because VLAN minimization is NP-hard, we employ a greedy VLAN-packing heuristic, Algorithm 2. Given the set of all paths P computed in Algorithm 1, Algorithm 2 processes the paths serially, constructing a set of subgraphs SG that include those paths. For each path p, if p is not covered by any subgraph in the current set SG, the algorithm tries to greedily pack that path p into any one of the subgraphs in the current set (lines 10–13). If the greedy packing step fails for a path, a new graph is created with this path, and added to SG (lines 15–18).

Running this algorithm just once might not yield a solution near the optimum. Therefore, we use the best solution from N runs, randomizing the order in which paths are chosen for packing, and the order in which the current set of subgraphs SG are examined.

5.3.3 An example

Fig. 6 shows a relatively simple wiring topology with seven switches. One can think of this as a 1-level tree (with switch #1 as the root), augmented by adding three cross-connect links to each non-root switch.

Algorithm 2 Greedy VLAN Packing Heuristic

```
 1: Given: G = (V, E), k
 2: SG = /* set of loop-free subgraphs*/
 3: for v ∈ V do
 4:   for u ∈ V do
 5:     P =ComputePaths(G, v, u, k) ;
 6:     for p ∈ P /* in a random order */ do
 7:       if p not covered by any graph in SG then
 8:         Success = FALSE;
 9:         for S ∈ SG /* in a random order */ do
10:           if p does not create loop in S then
11:             Add p to S
12:             Success = TRUE ;
13:           end if
14:         end for
15:         if Success == FALSE then
16:           S' = new graph with p
17:           SG = SG ⋃ {S'}
18:         end if
19:       end if
20:     end for
21:   end for
22: end for
23: return SG
```

Fig. 7 shows how the heuristic greedy algorithm (Alg. 2) chooses seven VLANs to cover this topology. VLAN #1 is the original tree (and is used as the default spanning tree).

5.4 Parallel Algorithm

The serial nature of Algorithm 2 does not scale well. We can parallelize VLAN computation on a node-by-node basis (where a "node" is an edge switch), using an algorithm based on graph-coloring heuristics. For each node in the input graph, an instance of the parallel algorithm computes the paths from all other nodes to the given node, and computes the minimum number of

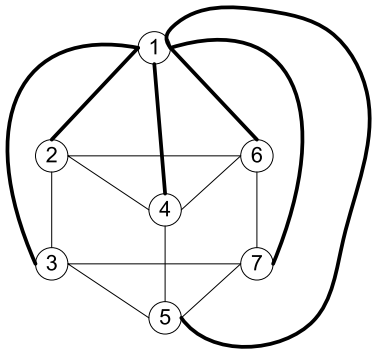


Figure 6: A seven-switch topology. Highlighted links show the original tree.

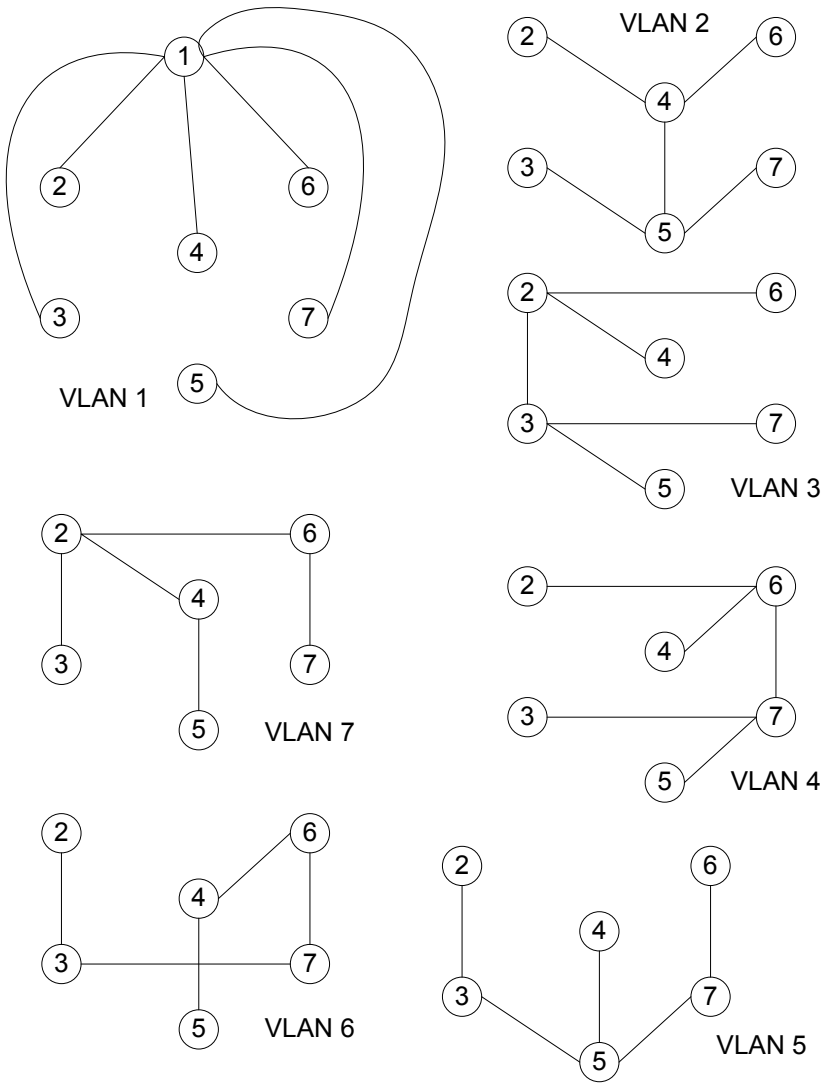


Figure 7: VLANs covering the seven-switch topology of Fig. 6, found by Alg. 2 (greedy VLAN-packing)

VLANs required for laying out those paths. There are no dependencies between these instances, so one can expect linear speedup in the number of edge switches.

Note that the problem of computing VLANs for a given destination node in a graph is essentially the restricted VLAN minimization problem, defined in Problem 2. Not only can we polynomially reduce the graph vertex coloring problem to the restricted VLAN Minimization problem, but we also observe that an instance of the restricted VLAN Minimization problem can be reduced back to an instance of the graph vertex-coloring problem. Several heuristics and approximation algorithms have been proposed for graph coloring [39]. We can leverage those techniques to compute a solution for a restricted VLAN minimization instance.

Algorithm 3 shows the per-destination VLAN computation algorithm. We first compute paths from all nodes to a given destination node d in the input graph, resulting in a path set P. We then construct a graph vertex coloring instance $G_{color} = (V_{color}, E_{color})$ for this set of paths as follows. For each path in P, we create vertex in G_{color}. For every two paths that are not *vlan-compatible*, we place an edge between the corresponding vertices in G_{color}. This ensures that those two paths are assigned different colors in a solution for the vertex coloring problem. We run a simple greedy vertex-coloring heuris-

Algorithm 3 Per destination VLAN computation

Given: Topology Graph $G = (V, E)$, number of paths k, destination d
Pathset $P = \emptyset$
for $v \in V$ **do**
  $P = P \bigcup$ComputePaths(G, v, d, k) ;
end for
$V_{color} = \{v_1, v_2, ..., v_{|P|}\}$; $E_{color} = \emptyset$;
for $p_i, p_j \in P$ **do**
  if *vlan-compatible*(p_i, p_j) = FALSE **then**
    $E_{color} = E_{color} \bigcup \{v_i, v_j\}$
  end if
end for
(k, f) =*vertex-coloring*$(G_{color} = (V_{color}, E_{color}))$
/* f maps each vertex v_i (and hence path p_i) to one of k colors */
SG_d =;
for $1 \leq i \leq k$ **do**
  G_i =MergePaths$(\{p_j \in P : f(v_j) = i\})$;
  $SG_d = SG_d \bigcup G_i$
end for
return SG_d

tic [39] on this graph coloring instance, which returns a number k representing the number of colors needed for coloring the input graph, and a function f mapping each vertex in the input graph to one of the colors. After running the vertex-coloring heuristics on G_{color}, we merge each subset paths with a single color into a subgraph. Thus, the algorithm returns a set of subgraphs corresponding to the VLANs laid out for the given destination.

With this approach, the number of VLANs needed will be $\sum_{d \in D} |SG_d|$, where SG_d is the set of subgraphs returned by the Algorithm 3 for a destination d. Remember that we can have at most 4096 VLANs. In a network with a few thousand switches, this summation can easily exceed 4096. So, we use a greedy merging heuristic to further reduce the number of VLANs required.

Subgraph Merging: After computing the set of subgraphs for each destination, we try to merge the subgraphs of different destinations, to reduce the overall number of VLANs required. Note that any two subgraphs computed by Algorithm 3 for two different destinations can be merged into a single graph, if and only if such merging causes no loops in the combined graph.

Algorithm 4 describes our heuristic to merge as many subgraphs as possible. The input to this routine is the union $\bigcup_{d \in V} SG_d$, where V and SG_d are as defined in Algorithm 3. In plain words, the input is the set comprising all subgraphs computed for all destinations using Algorithm 3. For each graph in the input, this algorithm tries to combine that graph with as many other graphs in the input as possible. The subroutine `isThereALoop`

Algorithm 4 Algorithm for Merging VLANs

Given: A set of VLAN Graphs $G = \{G_1, G_2, ...G_n\}$
$G' =$
for $G_i \in G$ **do**
  $G = G - \{G_i\}$
  for $G_j \in G$ **do**
    $G_m = G_i \bigcup G_j$
    if isThereALoop(G_m) == FALSE **then**
      $G = G - \{G_j\}$
      $G_i = G_m$
    end if
  end for
  $G' = G' \bigcup \{G_i\}$
end for
return G'

used in this Algorithm runs a breadth-first search on the input graph, to check for the presence of a loop. In effect, the algorithm considers candidate subgraphs in some order, does a trial merge of one subgraph into the current merged subset, and accepts the result if it is loop-free.

Note that the order in which the graphs are combined will affect the size of the final result. For our implementation, we have taken the simple approach of considering the graphs in a random order.

5.5 Algorithm performance

Since our algorithm is an approximate solution to an NP-hard problem, we applied it to a variety of different topologies that have been suggested for data-center networks, to see how many VLANs it requires. Where possible, we also present the optimal number of VLANs.

We considered the following topologies:

- FatTree(p) [10]: This topology, a 2-ary 3-tree, is parametrized on p, where p is the number of ports per switch. This creates a full bisection bandwidth network for $p^3/4$ end hosts. Between any switches, there are $(p/2)^2$ edge-disjoint paths.
- BCube(p, l) [20]: This topology is parametrized on two variables p and l, where p is the number of ports per switch used in the topology and l is the number of levels in the recursive construction of the topology. BCube is presented as a server-centric architecture, where each server has l NICs and also acts as a switch. For this paper, we simply redraw each of the server/switch combinations as an edge switch with several end-host servers connected. Between any two switches, there are l edge-disjoint paths.
- 2-D HyperX(k) [9]: This topology is parametrized on k, where k is the number of switches in each dimension of a 2-D mesh. Each switch is connected directly to all switches in the same row and the same

column. Hence, between any two switches, there are $2(k-1)$ edge-disjoint paths.

- CiscoDC(m, a): Cisco's recommended data center network [11]. This is a three-layer topology with two core switches at the top, several pairs of aggregation level switches (each pair referred to as an aggregation module), and several pairs of access switches associated with each aggregation module. Switches in each access-switch pair have a link between them, and are also connected to both the switches of their aggregation module. We parametrize this topology on two variables m and a, representing the number of aggregation modules and the number of access switch pairs associated with each aggregation module. There are $2ma$ access switches and 3 edge-disjoint paths between any two access switches in this network.

Table 1: Performance of VLAN mapping heuristic

Topology	**Minimal # of VLANs**	**SPAIN's heuristic**	**Trials (N) for best**
FatTree (p)	$(p/2)^2$	$(p/2)^2$	1 for all p
BCube (p, l)	$p^{l-1}l$	$p^{l-1}l$	290 for (2,3) 6 for (3,2)
2-D HyperX (k)	Unknown $O(k^3)$	12 for k=3 38 for k=4	475 304
CiscoDC (m, a)	Unknown	9 for (2,2) 12 for (3,2) 18 for (4,3)	1549 52 39
our testbed	4	4	4
Fig. 6	7	7	137

Table 1 shows the performance of SPAIN's VLAN mapping heuristic on different topologies. The heuristic matches the optimal mapping on FatTree and BCube. We don't yet know the optimal value for CiscoDC or 2-D HyperX, although for 2-D HyperX, k^3 is a loose upper bound. The table also shows that, for the Open Cirrus subset used in our experiments (Sec. 10), the heuristic uses the optimal number (4) of VLANs.

The above strategy is quite successful on FatTree and BCube topologies. For FatTree, our heuristic automatically determines the best VLAN layout as the set of spanning trees rooted at each of the root switch. Thus with $(k/2)^2$ VLANs, all $(k/2)^2$ paths between any two edge switches are covered. For BCube, our algorithm returns the set of spanning trees each rooted at a non-edge switch. Thus with $n^{k-1}k$ VLANs, we can cover all k paths for any pair of edge switches.

The last column in the table shows the number of trials (N) it took for SPAIN's VLAN packing algorithm to generate its best result; we show the worst case over five runs, and the averages are much smaller. In some cases, luck seems to play a role in how many trials are required. Each row took less than 60 sec., using a single CPU (for these computations, we used the serial algorithm, not the parallel algorithm).

5.6 Fault tolerance in SPAIN

A SPAIN-based network must disable the normal STP behavior on all switches; otherwise, they will block the use of their non-spanning-tree ports, preventing SPAIN from using those links in its VLANs. (SPAIN configures its VLANs to avoid loops, of course.) Disabling STP means that we lose its automatic fault tolerance.

Instead, SPAIN's fault tolerance is based on the pre-provisioning of multiple paths between pairs of hosts, and on end-host detection and recovery from link and switch failures; see Sec. 6.6 for details.

However, SPAIN could use features like Cisco's proprietary Per-VLAN Spanning Tree (PVST) or the IEEE 802.1s standard Multiple Spanning Tree (MST) to improve fault tolerance. SPAIN could configure switches so that, for each VLAN, PVST or MST would prefer the ports in that VLAN over other ports (using per port spanning tree priorities or weights). This allows a switch to fail over to the secondary ports if PVST or MST detects a failure. SPAIN would still use its end-host failure mechanisms for rapid repair of flows as the spanning tree protocols have higher convergence time, and in case some switches do not support PVST/MST.

5.7 FIB pinning

The design of SPAIN presented so far depends on switches to initially learn the locations of MAC addresses in the topology. A learning switch, for every packet received on an interface i with a source MAC-address m_{src} on VLAN v, records an entry $\langle v, m_{src}\rangle \rightarrow i$. Thus, the switch learns that address m_{src} can be reached via interface i for VLAN v. It records this in its Forwarding Information Base (FIB) table. A switch's FIB table maintains mappings from a (VLAN, MAC-address) pair to an outgoing interface.

When a switch receives a packet with destination MAC address that it has not yet learned, then it floods that packet on all interfaces except the one on which it has received the packet. Thus, the learning process can create overheads from the resultant packet-flooding; as we describe in Sec. 6, SPAIN's end-host algorithms normally includes a *chirping* protocol, which significantly reduces flooding due to switch table timeouts.

If we want to avoid the need for learning, flooding, and the chirping protocol, we can implement SPAIN instead using a variant called *FIB pinning*. FIB pinning leverages our ability to directly program the FIB tables on the switches via SNMP, and assumes that a central controller knows the MAC addresses and locations of

each NIC (real or virtual) in the network. In FIB pinning, SPAIN not only sets up the VLAN-map table, it also pre-configures switches so that they know which hosts are on each VLAN, and where.

In the basic SPAIN approach, we pre-configure each switch to know which of its links belong to each VLAN, and we ensure that each VLAN is loop-free; the switches can then use flooding to learn the path to each unknown destination MAC address on each VLAN. In the FIB-pinning approach, we pre-configure the switch FIBs to contain next-hop entries for each $\langle v, m_{dest} \rangle \rightarrow i$ pair, so no flooding or learning is necessary. This also means that we can share VLANs between per-destination trees, reducing the number of VLANs required for a given topology.

This VLAN sharing can create loops within VLANs. As long as an end host follows SPAIN's rules for selecting the right VLAN to reach a specific destination, its packets will be forwarded without looping. However, if a non-SPAIN host tries to use one of these VLANs, this can led to an infinite forwarding loop, so the switch ports connected to non-SPAIN hosts must be configured to disallow the use of SPAIN's VLANs. (Also, a buggy SPAIN host could lead to looping packets, by sending packets to a non-SPAIN host on one of these VLANs.)

Because FIB pinning requires central knowledge of the location of MAC addresses, it is therefore not as widely applicable as the learning-based approach. It requires a highly-available online central controller that can respond to link failures by changing the FIB entries. It may also create problems for VM migration, since FIB entries in all switches must be updated in sync with a VM's change of physical location within the network. Finally, a major disadvantage of FIB pinning is that it can require a very large number of FIB entries. Even when no host connected to an edge-switch ever communicates with a destination d, the FIB of that edge switch must still be populated with entries for d, because we have no way to know *a priori* that such communication will never take place.

Thus, a key trade-off between the basic learning-based approach and FIB pinning is that the former uses some extra bandwidth due to flooding on learning misses, while the latter may require large FIB tables.

5.7.1 FIB-pinning algorithm

To pin a path from a source host s to a destination host d, we can assign a VLAN v to that path, and populate an entry for $\langle v, d \rangle$ in the FIB for each switch on that path.

Note that reducing the number of VLANs is important, even if we can use a large number of VLANs. If two paths to a destination d go through a switch s, assigning two different VLANs to these paths requires two different FIB entries in the switch s. In contrast, only one FIB entry is required if we can assign a single VLAN to both these paths.

This is important, because switch FIB table sizes are limited. Typical switches have 16K FIB entries on their fast datapath (on-chip SRAM) and 128K FIB entries, in off-chip DRAM, on a slower datapath. A switch that can re-learn the locations of MAC addresses can handle, at some performance cost due to re-flooding, more addresses than will fit into its FIB table. But with FIB pinning, there is no way to support more (VLAN, MAC) pairs than that can be fit simultaneously into the FIB table of the smallest switch in the network.

Note that we can assign the VLAN number for multiple paths to a single destination d, if all of these paths are *vlan-compatible*. Hence, we can reuse the per-destination VLAN computation algorithm (Algorithm 3). As mentioned before, we can reuse the VLAN numbers across destinations; hence, we do not need to run the merging algorithm (Algorithm 4).

We present our algorithm for FIB Pinning in Algorithm 5. This algorithm first creates a corresponding vertex-coloring problem instance, and uses its solution for determining VLANs, similar to the Algorithm 5. The

Algorithm 5 Algorithm for assigning VLANs and generating FIB entries in the FIB Pinning Approach

```
1: Given: Topology Graph G = (V, E), number of paths k, destination d
2: Pathset P = ∅
3: for v ∈ V do
4:    P = P ⋃ ComputePaths(G, v, d, k) ;
5: end for
6: V_color = {v_1, v_2, ..., v_|P|}; E_color = ∅;
7: for p_i, p_j ∈ P do
8:    if vlan-compatible(p_i, p_j) = FALSE then
9:       E_color = E_color ⋃ {v_i, v_j}
10:   end if
11: end for
12: (k, f) = vertex-coloring(G_color = (V_color, E_color))
13: /* f maps each vertex v_i (and hence path p_i) to one of k colors */
14: for 1 ≤ i ≤ n do
15:   for 1 ≤ j ≤ |p_i| do
16:      /* Add a FIB entry to switch p_i^j for each machine at edge-switch d */
17:      for machine m at switch d do
18:         On switch p_i^j, add FIB entry: ⟨f(i), m⟩ → int(p_i^{j+1}, p_i^j)
19:         /* Function int(s_2, s_1) returns interface on s_1 that connects to s_2 */
20:      end for
21:   end for
22: end for
```

algorithm then fills in each switch's FIB table on each path. We invoke Algorithm 5 separately for each destination.

Overall, the number of VLANs $|V|$ required with the FIB-pinning approach is the maximum of the number of VLANs required for any destination. If D is the set of destinations in our topology, and k_d is the number of VLANs required for a destination d, then

$$|V| = \max_{d \in D} k_d. \quad (4)$$

The number of FIB entries needed on a switch is the sum of number of unique VLANs for each destination. Assuming P_d is the set of paths to a destination d, m_d is the number of end hosts attached to that destination edge-switch d, and f_d maps a path for destination d to a VLAN number, the number of FIB entries F_s needed on a switch s is

$$F_s = \sum_{d \in D} m_d \times | \bigcup_{p \in P_d, s \in p} \{f_d(p)\}| \quad (5)$$

Note that F_s grows both with the number of total end hosts in the network and with the number of VLANs used. In Sec. 8.2, we study the number of FIB entries needed for topologies of various sizes.

6 End-host algorithms

Once the off-line algorithms have computed the paths and configured the switches with the appropriate VLANs, all of the online intelligence in SPAIN lies in the end hosts.[2] SPAIN's end-host algorithms are designed to meet five goals: (1) effectively spread load across the pre-computed paths, (2) minimize the overheads of broadcasting and flooding, (3) efficiently detect and react to failures in the network, (4) facilitate end-point mobility (e.g., VM migration), and (5) enable incremental deployment. These algorithms can be implemented in several different ways, as described in Sec. 9, none of which require significant changes to the end-host OS or hypervisor. We generically refer to the implementation as the "SPAIN driver," although it need not be an actual device driver.

The SPAIN driver has four major functions: boot-time initialization, sending a packet, receiving a packet, and re-initializing a host after it moves to a new edge switch.

An end host uses the following data structures and parameters:

- $ES(m)$: the ID of the edge switch to which MAC address m is currently connected.
- $V_{reach}(es)$: the set of VLANs that reach the edge switch es.
- R: the *reachability* VLAN map, a bit map encoding the union of $V_{reach}(\bullet)$ over all es, computed by the algorithms in Section 5.
- $V_{usable}(es)$: the set of VLANs that have recently tested as usable to reach es.
- T_{repin} is the length of time after which non-TCP flows go through the VLAN re-pinning process.
- T_{sent} is the minimum amount of time since last send on a VLAN that triggers a *chirp* (see below).
- $V_{sent}(es)$: the set of VLANs that we sent a packet via es within the last T_{sent} seconds.

SPAIN uses a protocol we calling *chirping* for several functions. A host sends a *chirp* packet with the triple <IP Address, MAC address, Edge-switch ID>. Chirp packets also carry a $want\_reply$ flag to trigger a unicast chirp in response; broadcasts chirp never set this flag. All hosts that receive a chirp update their ARP tables with this $IP \rightarrow MAC$ address binding; they also update the $ES(m)$ table. SPAIN sends unicast chirps often enough to preempt most of the flooding that would arise from entries timing out of switch learning tables.

6.1 Host initialization

After a host boots and initializes its NIC drivers, SPAIN must do some initialization. The first step is to download the VLAN reachability map R from a repository. (The repository could be found via a new DHCP option.) While this map could be moderately large (about 5MB for a huge network with 500K hosts and 10K edge switches using all 4K possible VLANs), it is compressible and cachable, and since it changes rarely, a re-download could exploit differential update codings.

Next, the driver determines the ID of the edge switch to which it is connected, by listening for Link Layer Discovery Protocol (LLDP) messages, which switches periodically send on each port. The LLDP rate (typically, once per 30 sec.) is low enough to avoid significant end-host loads, but fast enough that a SPAIN driver that listens for LLDP messages in parallel with other host-booting steps should not suffer much delay.

Finally, the host broadcasts a chirp packet, on the default VLAN (VLAN 1). Although broadcasts are unreliable, if host Y fails to receive a broadcast chirp from host X, it Y will later recover by sending a unicast chirp (with the $wants\_response$ flag set) when it needs to select a VLAN for communicating with host X.

6.2 Sending a Packet

SPAIN achieves high bisection bandwidth by spreading traffic across multiple VLANs. The SPAIN driver must choose which VLAN to use for each flow (we normally avoid changing VLANs during a flow, to limit packet reordering). Therefore, the driver must decide which VLAN to use when a flow starts, and must also decide whether to change VLANs (for reasons such as routing around a fault, or improving load-balance for long

[2] SPAIN could support the use of a centralized service to help end-hosts optimize their load balancing, but we have not yet implemented this service, nor is it a necessary feature.

Algorithm 6 Selecting a VLAN

```
1: /* determine the edge switch of the destination */
2: m = get_dest_mac(flow)
3: es = get_es(m)
4: /* candidate VLANs: those that reach es */
5: if candidate_vlans is empty then
6:    /* No candidate VLANs; */
7:    /* Either es is on a different SPAIN cloud or m is a non-
      SPAIN host */
8:    return the default VLAN (VLAN 1)
9: end if
10: /* see if any of the candidates are usable */
11: usable_vlans = candidate_vlans ⋂ V_usable(es)
12: if usable_vlans is empty then
13:    return the default VLAN (VLAN 1)
14: end if
15: init_probe(candidate_vlans − usable_vlans)
16: return a random V ∈ usable_vlans.
```

flows, or to support VM mobility). We divide these into two algorithms: for VLAN selection, and for triggering re-selection of the VLAN for a flow (which we call *re-pinning*).

Algorithm 6 shows the procedure for VLAN selection, given a destination MAC address m. The driver uses the $ES(m)$ table (maintained by chirps) to find the edge switch es for the destination, and then uses the Reachability Map R to find the set of VLANs that reach es. (Non-SPAIN hosts do not appear in $ES(m)$, and so are reached via the default VLAN.) The driver then computes the *candidate set* by removing VLANs that are not in $V_{usable}(es)$ (which is updated during processing of incoming packets; see Algorithm 8).

If the candidate set is non-empty, the driver selects a member at random and uses this VLAN for the flow.[3] If the set is empty (there are no known-usable VLANs), the flow is instead assigned to VLAN 1. The driver initiates *probing* of a subset of all the VLANs that reach the es but are currently not usable.

SPAIN probes a VLAN V to determine whether it can be used to reach a given destination MAC m (or its ES) by sending a unicast chirp message to m on V. If the path through VLAN V is usable and if the chirp reaches m, the receiving SPAIN driver responds with its own unicast chirp message on V, which in turn results in V being marked as in the probing host (the bit $V_{usable}(es)$ is set to 1).

When to re-pin?: Occasionally, SPAIN must change the VLAN assigned to a flow, or *re-pin* the flow. Repinning helps to solve several problems:

1. Fault tolerance: when a VLAN fails (that is, a link or switch on the VLAN fails), SPAIN must rapidly move the flow to a usable VLAN, if one is available.
2. VM migration: if a VM migrates to a new edge switch, SPAIN may have to re-assign the flow to a VLAN that reaches that switch
3. Improving load balance: in the absence of an online global controller to optimize the assignment of flows to VLANs, it might be useful to shift a long-lived flow between VLANs at intervals, so as to avoid pathological congestion accidents for the entire lifetime of a flow.
4. Better VLAN probing: the re-pinning process causes VLAN probing, which can detect that a "down" VLAN has come back up, allowing SPAIN to exploit the revived VLAN for better load balance and resilience.

When SPAIN detects either of the first two conditions, it immediately initiates re-pinning for the affected flows.

However, re-pinning for the last two reasons should not be done *too* frequently, since this causes problems of its own, especially for TCP flows: packet reordering, and (if re-pinning changes the available bandwidth for a flow) TCP slow-start effects. Hence, the SPAIN driver distinguishes between TCP and non-TCP flows. For non-TCP flows, SPAIN attempts re-pinning at regular intervals.

For TCP flows, re-pinning is done only to address failure or serious performance problems. SPAIN initiates re-pinning for these flows only when the congestion window has become quite small, and the current (outgoing) packet is a retransmission. Together, these two conditions ensure that we do not interfere with TCP's own probing for available bandwidth, and also eliminate the possibility of packet reordering.

Algorithm 7 illustrates the decision process for repinning a flow; it is invoked whenever the flow attempts to send a packet.

6.3 Receiving a Packet

Algorithm 8 shows pseudo-code for SPAIN's packet reception processing. All chirp packets are processed to update the host's ARP table and ES table (which maps MAC addresses to edge switches); if the chirp packet requests a response, SPAIN replies with its own unicast chirp on the same VLAN.

If the packet is a broadcast or multicast packet, it is simply sent up to the higher protocols, without further processing by the SPAIN driver.

The driver treats any incoming packet (including chirps) as proof of the health of the path to its source edge switch es via the arrival VLAN.[4] It records this observation in the $V_{usable}(es)$ bitmap, for use by Algorithm 6.

Finally, before delivering the received packet to the

[3] SPAIN with a dynamic centralized controller could bias this choice to improve global load balance; see Sec. 9.

[4] In the case of an asymmetrical failure in which our host's packets are lost, SPAIN will ultimately declare the path dead after our peer gives up on the path and stops using it to send chirps to us.

Algorithm 7 Determine if a flow needs VLAN selection

```
1: if last_move_time >= last_pin_time then
2:    /* we moved since last VLAN selection - re-pin flow */
3:    return true;
4: end if
5: current_es = get_es(dst_mac)
6: if saved_es (from the flow state) ! = current_es then
7:    /* destination moved – update flow state & re-pin */
8:    saved_es = current_es;
9:    return true
10: end if
11: if current_vlan(flow) ≤ 0 then
12:    return true /* new flows need VLAN selection */
13: end if
14: if proto_of(flow) ≠ TCP then
15:    if (now − last_pin_time) ≥ T_repin then
16:       return true /* periodic re-pin */
17:    end if
18: else
19:    if cwnd(flow) ≤ W_repin_thresh && is_rxmt(flow)
       then
20:       return true /* TCP flow might prefer another path */
21:    end if
22: end if
23: return false /* no need to repin */
```

protocol stack, SPAIN sends a unicast chirp to the source host if one has not been sent recently. (The pseudo-code omits a few details, including the case where the mapping $ES(mac)$ is unknown. The code also omits details of deciding which chirps should request a chirp in response.)

6.4 Table housekeeping

The SPAIN driver must do some housekeeping functions to maintain some of its tables. First, every time a packet is sent, SPAIN sets the corresponding VLAN's bit in $V_{sent}(es)$.

Periodically, the $V_{sent}(es)$ and $V_{usable}(es)$ tables must be cleared, at intervals of T_{sent} seconds. To avoid chirp storms, it might be a good idea to perform these table-clearing steps in evenly-spaced chunks, rather than clearing the entire table at once.

6.5 Support for end-host mobility

SPAIN makes a host that moves (e.g., for VM migration) responsible for informing all other hosts about its new location. After a host has finished its migration, it broadcasts a chirp, which causes the recipient hosts to update their ARP and ES tables. Since a partial failure of a VLAN might only affect the new location, the migrating host should also flush its $V_{usable}(es)$ table.

6.6 Handling failures

Failure detection, for a SPAIN end host, consists of detecting a VLAN failure and selecting a new VLAN for the affected flows; we have already described VLAN selection (Algorithm 6).

Algorithm 8 Receiving a Packet

```
1: vlan = get_vlan(packet)
2: m = get_src_mac(packet)
3: if is_chirp(packet) then
4:    update_ARP_table(packet)
5:    update_ES_table(packet, vlan)
6:    if wants_chirp_response(packet) then
7:       send_unicast_chirp(mac, vlan)
8:    end if
9: end if
10: if not_unicast(packet) then
11:    deliver packet to protocol stack
12: end if
13: es = get_es(m) /* determine sender's edge switch */
14: /* mark packet-arrival VLAN as usable for es */
15: V_usable(es) = V_usable(es) ⋃ vlan
16: /* chirp if we haven't sent to es via vlan recently */
17: if the vlan bit in V_sent(es) is not set then
18:    send_unicast_chirp(mac, vlan)
19:    /* V_sent(es) is cleared every T_sent sec. */
20: end if
21: deliver packet to protocol stack
```

While we do not have a formal proof, we believe that SPAIN can almost always detect that a VLAN has failed with respect to an edge switch es, because most failures result in observable symptoms, such as a lack of incoming packets (including chirp responses) from es, or from severe losses on TCP flows to hosts on es.

SPAIN's design improves the chances for rapid failure detection because it treats all received packets as probes (to update V_{usable}), and because it aggregates path-health information per edge switch, rather than per destination host. However, because switch or link failures usually do not fully break an entire VLAN, SPAIN does not discard an entire VLAN upon failure detection; it just stops using that VLAN for the affected edge switch(es).

SPAIN also responds rapidly to fault repairs; the receipt of any packet from a host connected to an edge switch will re-establish the relevant VLAN as a valid choice. SPAIN also initiates re-probing of a failed VLAN if a flow that could have used the VLAN is either starting or being re-pinned. At other times, SPAIN re-probes less aggressively, to avoid unnecessary network overhead.

6.7 End-host-based load balancing

To this point, we have deferred discussing how an end host chooses between multiple usable paths. This choice, however, can dramatically affect the performance of a multi-path data center network, because it is the primary mechanism for dynamic load balancing.

SPAIN does not depend on a specific load-balancing scheme, but allows the use of a wide variety, including both centralized and decentralized approaches. Except in the case (described in Sec. 6.6) where a path fail-

ure causes VLAN re-selection, SPAIN chooses a VLAN only at the start of a flow.

Techniques that can be implemented entirely locally at each end host include randomly choosing a VLAN for each flow; cycling among the available VLANs (round-robin); or hashing on some part of the flow address tuple so as to spread load according to a static policy.

Alternatively, SPAIN can exploit an *optional* online centralized controller, to provide global management of network load. In this model, the global controller (which could be implemented as a distributed system, for availability and performance) monitors the links in the network to detect significant load imbalances, and causes end hosts to choose VLANs so as to shift load away from overloaded links. (Similarly, the NOX network operating system controller for OpenFlow "can utilize real-time information about network load and the utilization of switches to install flows on uncongested links" [38].)

One plausible approach, which we have implemented in our prototype (see Sec. 9.1), is for the centralized controller to augment the reachability map R with weights for each VLAN. The end host's selection algorithm (e.g., random, round-robin, or hash-based) then biases its VLAN selection based on these weights. The map can be updated periodically, at a rate low enough to avoid much network load, but fast enough to respond to changing conditions. If update is lost or the controller fails, the SPAIN end-host behavior is sufficiently autonomous to allow communication to continue, albeit with some load imbalance.

A possible variation on this approach is for the controller to deliver differently-weighted versions of R to each end-host, although we are not sure this is necessary to achieve good load balance.

7 How SPAIN meets its goals

In this section, we discuss how the design of SPAIN addresses the major goals we described in Sec. 4.

Efficiently exploit multiple paths in arbitrary topologies: SPAIN's use of multiple VLANs allows it to spread load over all physical links in the network, not just those on a single spanning tree. SPAIN's use of end-host techniques to spread load over the available VLANs also contributes to this efficiency.

Support COTS Ethernet switches: SPAIN requires only standard features from Ethernet switches. Also, because SPAIN does not require routing all non-local traffic through a single core switch, it avoids the need for expensive switches with high port counts or high aggregate bandwidths.

Tolerate faults: SPAIN pre-computes multiple paths through the network, so that when a path fails, it can immediately switch flows to alternate paths. Also, by avoiding the need for expensive core switches, it decreases the need to replicate expensive components, or to rely on a single component for a large subset of paths.

SPAIN constantly checks path quality (through active probing, monitoring incoming packets, and monitoring the TCP congestion window), thereby allowing it to rapidly detect path failures.

Support incremental deployment: None of SPAIN's end-host processing depends on an assumption that all end hosts implement SPAIN. (Our experiments in Sec. 10.4, showing the performance of SPAIN in incremental deployments, did not require any changes to either the SPAIN code or the non-SPAIN hosts.) Traffic to and from non-SPAIN hosts automatically follows the default VLAN, because these hosts never send chirp messages and so the SPAIN hosts never update their $ES(m)$ maps for these hosts.

8 Simulation results

We first evaluate SPAIN using simulations of a variety of network topologies. Later, in Sec. 10, we will show experimental measurements using a specific topology, but simulations are the only feasible way to explore a broader set of network topologies and scales.

We use simulations to (i) show how SPAIN increases link coverage and potential reliability; (ii) quantify the switch-resource requirements for SPAIN's VLAN-based approach; and (iii) show how SPAIN increases the potential aggregate throughput for a network.

We simulated both regular topologies (those generated by a rule) and arbitrary topologies. For regular topologies, we used several fat-tree designs ($p = 4, 8, 16, 48$, refer to Section 5.5). For a source of arbitrary topologies, we used some AS-level topologies from the RocketFuel project [37]. These are, of course, not data-center networks, but we use them (for lack of a ready source of enterprise network topologies) to illustrate the broad capabilities of SPAIN. For these AS-level topologies, we assume 50 end machines per each node in the topology. For a fat-tree topology, the port count p of the switches determines the topology and the number of end hosts that can be connected. With 48-port switches, FatTree(48) connects 27648 end-machines while providing full bisection bandwidth.

In this section, we refer to the desired number of paths per source-destination pair in a topology as the *Path-Set* size PS. FatTree(p) has $X = (p/2)^2$ edge-disjoint paths between any source-destination pair; hence, we use $PS = X$ as the path-set size for the simulations on Fat-Trees. Since RocketFuel (RF) topologies are arbitrary, we consider different path-set sizes varying from 1 (no guaranteed redundancy) to 3.

Table 2 shows properties of the topologies we evaluated, and summarizes the results of our simulations. In this table, for the RF topologies, we include results only

for a path-set size of 3.

The table shows the Autonomous System (AS) number for each of the RF topologies (T1–T6), and the number of switches, links, and hosts in each simulated topology.

As described in Section 5.2, we perform a graph compaction step, to reduce the working topology size for path computation and path layout phases. The F_v and F_e columns show the reduction factors in the number of vertices and edges, respectively, achieved by the graph-compaction step in Algorithm 1. Because our path computation and path layout algorithms are of $O(V^2(V + E))$ time complexity, these reduction factors (of up to 8.6) result in a large reduction in execution times of those phases.

The *Coverage* column shows the fraction of links covered by a spanning tree (SPAIN always covers 100% of the links, if enough VLANs are available.)

The *NCP* (no-connectivity pairs) column is indicative of the fault tolerance of a SPAIN network; it shows the expected fraction of source-destination pairs that lack connectivity, with a simulated link-failure probability of 0.04, averaged over 10 randomized trials. (These are for $PS = 3$; even for $PS = 1$, SPAIN would be somewhat more fault-tolerant than STP.)

The *VLANs* column shows the number of VLANs required for each topology; note that for T5, the number exceeds Ethernet's 4K limit, but we can support T5 with $PS = 2$, or with 94% coverage at $PS = 3$.

The *Throughput* column shows the aggregate throughput possible through the network, assuming unit per-link and per-sender throughputs, and fair sharing of links between flows. We assume that SPAIN chooses at random from the three available paths, and we report the mean of 10 randomized trials, normalized so that STP = 1.

In summary, SPAIN's paths cover more than twice the links, and with more than three times the reliability, of spanning-tree's paths. Even in most large, arbitrary topologies, SPAIN can find a VLAN assignment within Ethernet's 4K limit, and improves throughput over spanning tree.

In the rest of this section, we expand upon these summarized results.

8.1 Path Set Quality

We use two metrics to quantify the quality of paths that we compute: (i) Link Coverage: this metric tracks the fraction of links of a network covered by the VLANs, and (ii) Reliability: this metric tracks how many source-destination pairs can communicate under independent link failures, with link failure probability f. We compare these metrics against Spanning Tree, for various path-set sizes PS.

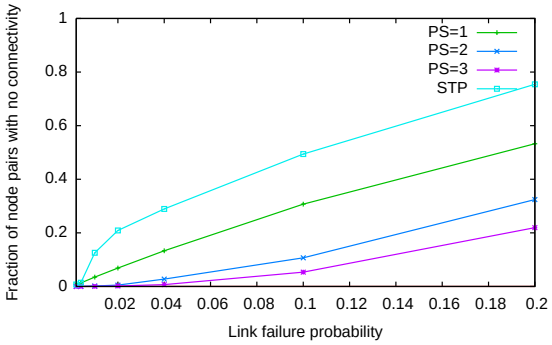


Figure 8: For different probabilities of link failure, comparison of reliability between SPAIN's paths and the Spanning Tree paths on topology T5: We plot the fraction of node pairs with no connectivity, which is also the failure probability of a randomly chosen path.

Link-Coverage For all topologies, and for the path-set sizes we simulated ($PS = 1..3$ for T1–T6, and $(p/2)^2$ for FatTree(p)), SPAIN achieves 100% link coverage.

Note that for smaller path-set sizes on fat-tree topologies, SPAIN will not achieve 100% link coverage, but will never be worse than Spanning Tree.

By contrast, Table 2 shows (in the *Coverage* column) that Spanning Tree can exploit only very few links (2%–37.5%) in the highly-redundant fat-tree topologies, and fewer than 52% of the links in the arbitrary topologies T1–T6.

Reliability For a given probability f of link failure, we compute the fraction NCP of source-destination pairs that loose connectivity in the Spanning Tree case and in the SPAIN case. NCP also denotes the failure probability of a randomly chosen path for the given link failure probability f. We performed 10 simulation trials, and report the averages in all of the following results.

In Figure 8, for topology T5, we plot the reliability NCP for different path set sizes, and compare them against Spanning Tree. Note that SPAIN's path sets have considerably better reliability in comparison to the Spanning Tree paths. We observed similar curves for other topologies. Table 3 presents this reliability metric across all topologies, for $f = 0.04$.

From these results, it is clear that SPAIN provides better reliability than Spanning Tree. The results show that larger path set sizes increase reliability, but even for $PS = 1$ in topologies T1–T6, SPAIN improves reliability over Spanning Tree.

8.2 Feasibility

We use the term "feasibility" to describe SPAIN's ability to find a VLAN mapping that fits within Ethernet's limit of 4096 VLANs. Table 4 shows number of VLANs

Topology	AS #	#Switches	#Links	#Hosts	F_v	F_e	Coverage	NCP (STP)	NCP (SPAIN)	VLANs	Throughput
FT(4)	–	20	32	16	1.0	1.0	37.5	35%	0%	4	2.00
FT(8)	–	80	256	128	1.0	1.0	15.62	17%	0%	16	4.00
FT(16)	–	320	2048	1024	1.0	1.0	7.03	21%	0%	64	8.00
FT(48)	–	2880	55296	27648	1.0	1.0	2.17	17%	0%	576	24.00
T1	4755	41	136	2050	1.3	1.2	51.72	16%	1%	198	1.69
T2	1755	295	1086	14750	1.6	1.3	41.95	23%	1%	2272	1.81
T3	3967	375	1698	18750	1.2	1.1	40.08	45%	2%	3153	2.12
T4	3257	411	1306	20550	2.2	1.5	43.62	26%	2%	1895	1.67
T5	3356	1620	13480	81000	2.2	1.2	12.57	29%	1%	5361	8.00
T6	1221	2532	6112	126600	8.2	3.7	36.93	30%	4%	2188	2.73

Key: F_v = vertex reduction factor; F_e = edge reduction factor; $Coverage$ = % of links covered by STP; *NCP*= % of node pairs with no connectivity, for link-failure probability = 0.04; $VLANs$ = # VLANs required; $Throughput$ = aggregate throughput, normalized to STP, for sufficient flows to saturate the network, except for T5 (see Sec. 8.3). (Path-set size = 3 for topologies T1–T6, and $(p/2)^2$ for FT(p) topologies)

Table 2: Summary of simulation results

Topology	NCP = failure prob. of a random path			
	STP	SPAIN		
		PS=1	PS=2	PS=3
T1	0.16	0.11	0.02	0.01
T2	0.23	0.17	0.04	0.01
T3	0.45	0.21	0.06	0.02
T4	0.26	0.20	0.04	0.02
T5	0.29	0.13	0.03	0.01
T6	0.30	0.18	0.06	0.04
	STP	PS = $(p/2)^2$		
FT(4)	0.35	0		
FT(8)	0.17	0		
FT(16)	0.21	0		
FT(48)	0.17	0		

Table 3: Reliability metric for different topologies at link failure probability of 0.04: Comparison between SPAIN's paths for different sizes (PS) and Spanning Tree Protocol (STP).

required for different topologies and for various path-set sizes. For fat-tree topologies, we use the same numbers presented in the Section 5.5. For topologies T1–T6, path-set sizes of 1 and 2 can be easily supported for all topologies, and a path-set size of 3 can be supported for a majority of them.

Column 3 of Table 4 shows the number of VLANs required based on the parallel algorithm described in Section 5.4 (Algorithm 3). For cases where this required 4000 or VLANs, we also ran Algorithm 4, to merge the VLANs of different destinations. The results, shown in column 4, show reductions of 37%, 22%, and 53% for the T3, T5, and T6 topologies respectively (for $PS = 3$). For the T5 topology, we further observe that, within the limit of 4096 VLANs, we could lay out 94% of the paths considered.

For the FIB-pinning approach (Sec. 5.7), Table 4 shows the number of VLANs and the number of FIB entries needed to implement each of the cases. Since VLAN numbers can be re-used across destinations, the number of VLANs required in this case is the maximum number of VLANs required for any one edge switch. Hence, we can implement all cases considered with a small number of VLANs.

However, the number of FIB entries required is proportional to the product of the number of end-host and the number of VLANs in a topology, and Table 4 shows that this number can become quite large. Typical switches support 16K FIB entries in fast SRAM and up to 128K entries in off-chip DRAMs. These limits imply that, with FIB pinning, we can set up a shortest path in all topologies T1–T6, and up to two paths in topologies with a few tens of thousands of end hosts.

8.3 Effectiveness

We use the term "effectiveness" to refer to the ability to support the maximum possible throughput for a given physical topology. We ran simulations to study this metric.

We used the following traffic model: To simulate N simultaneous flows, we uniformly randomly choose N source-destination pairs. For each flow, we assume that the sender has an infinite amount of data to send to the destination, and that it sends data at the rate of at most one data unit per unit time. Each link in our topology can transmit at most one unit of data per unit time. We assume that the bandwidth of a link is fairly distributed among all flows through that link. This means that each flow going over a link with m flows can transmit at $1/m$ rate, limited of course by the link at which it encounters the largest m.

We compute the aggregate throughput of all flows, measured in the number of units of data per unit time, for different numbers of flows. For each of these simulation experiments, we performed 10 trials and computed the average across those trials. For SPAIN with path-set sizes greater than 1, we simulate the case in which the source node chooses one of the paths randomly.

Table 4: Feasibility results

Topology	PS	Basic SPAIN		FIB Pinning	
		#V	#V' (P_c)	#V	#FIB
T1	1	31		1	2050
	2	130		5	8150
	3	198		8	12600
T2	1	181		1	14750
	2	1101		9	69650
	3	2272		18	148k
T3	1	322		1	18750
	2	2362		11	98900
	3	5022	3153 (100%)	27	221k
T4	1	189		1	20550
	2	1020		8	92550
	3	1895		14	169k
T5	1	737		1	81000
	2	3906		8	302k
	3	6900	5361 (94%)	15	527k
T6	1	310		1	126600
	2	2370		11	858k
	3	4696	2188 (100%)	23	1722k
FT(4)	4	4		4	64
FT(8)	16	16		16	2048
FT(16)	64	64		64	64K
FT(48)	576	576		576	16M

PS is the size of path set. #V corresponds to the number of VLANs required as computed by the parallel Algorithm 3. #V' corresponds to the number of VLANs required after we have run Algorithm 4, and P_c is the percentage of the path set that is covered by limiting the number VLANs to at most 4096. #FIB represents the number of FIB entries that that FIB pinning would require to realize the corresponding path-set size.

For the fat-tree topologies, the throughput results are straightforward, and are shown in Table 2 (for 10 million flows), and in Fig. 9. These results can be calculated by straightforward means, but we also confirmed our calculations via simulations.

For the arbitrary topologies (T1–T6), we had to simulate. Our simulation framework runs on a Java VM, on a server with 4 CPU cores and 16GB of RAM; this allows us to simulate up to 10 million flows. For topologies T1–T4 and T6, that was enough to saturate the entire network. For T5, however, we could not fully saturate the network with this simulation configuration.

In Figure 10, we plot, for each topology T1–T6, the aggregate throughput At a first look, these results are perplexing; somehow, having more paths between node pairs results in decreased aggregate throughput. Upon further analysis, we noted that at a high number of flows, the network becomes overloaded, with several flows between *every* source-destination pair. A flow from a source to a destination that takes a non-shortest path disrupts more flows than a flow that takes the shortest path, as the non-shortest path (by definition) traverses more links than the shortest path.

These simulation results suggest several possible improvements to SPAIN:

- Since the use of short paths helps overall network performance, an end host should prefer to use shorter paths over longer ones. The path lengths are known as a result of the algorithms that compute them, and this information could easily be provided to the end hosts, at the cost of slightly larger tables.
- The results for larger path-set sizes show that, at lower numbers of flows, they provide better overall network performance than smaller path-set sizes. An online global controller could exploit this information to help improve load balance.

However, we are not sure if these observations would apply to typical data-center topologies.

Table 5 summarizes the throughput results for 10 million flows. For topologies T1–T6, we observe at least 67% improvement (T4, $PS = 3$), and up to 10X improvement (T5, $PS = 1$). Our experiments with 10 million flows saturated the Spanning Tree links in case of topology T5 (see Figure 10, topology T5), but did not saturate the SPAIN cases.

Fat-tree topologies provide full bisection bandwidth by means of tremendous redundancy, so in these networks we see as many simultaneous full-speed flows as the number of end hosts. Spanning tree cannot leverage that redundancy, so, as expected, our results show up to 24X improvement with SPAIN on fat-tree topologies.

9 Linux end-host implementation

We considered three possible approaches to the end-host implementation for SPAIN in Linux:

- A custom kernel module.
- The "off-the-shelf" OpenFlow [25] kernel module (section 9.1).
- A user-mode-only implementation, with no kernel changes, based on the widely-available 802.1Q and bonding-driver Linux modules; this lacks chirping and fault-tolerance support (section 9.2).

A custom kernel module for SPAIN would probably be relatively easy to write, would support the fault-tolerance and chirping aspects of SPAIN that the other two versions do not, and would use less CPU and RAM, but this is future work.

9.1 Approach 1: OpenFlow kernel module

Our implementation of the SPAIN prototype uses the OpenFlow kernel module, which was chosen for

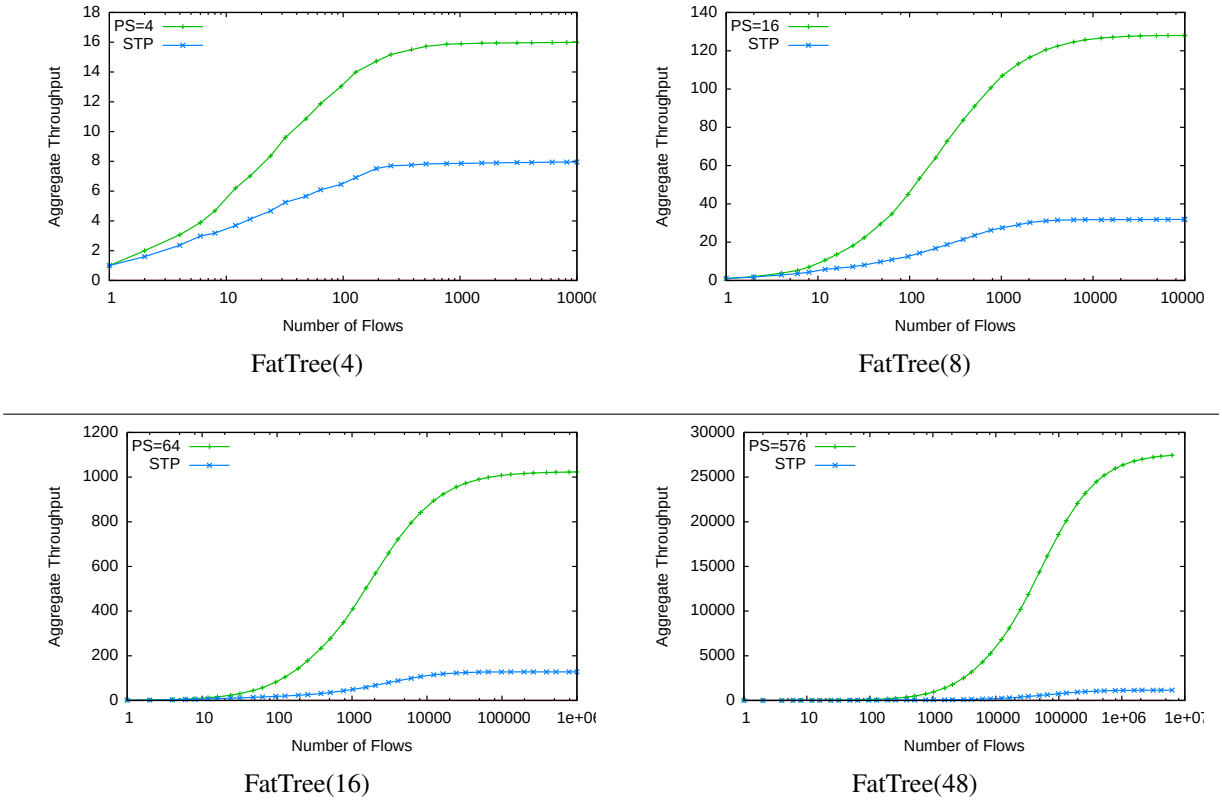


Figure 9: Aggregate throughput vs. number of flows: SPAIN and Spanning Tree on fat-tree topologies.

expediency, as it readily supports VLAN encapsulation/decapsulation. The OpenFlow protocol itself is not used in the network.

This approach does not support the failure-detection mechanisms described in Sec. 6.6. It also adds considerable CPU overhead (see Sec. 10.2).

We wrote a local, user-mode "OpenFlow controller" to implement the SPAIN mechanisms by manipulating the OpenFlow kernel module through its existing interface. The kernel module, local controller, and associated scripts amount to about 4MBytes that must be installed on each host.

This implementation does not currently use the chirping mechanism, relying instead on a central repository to provide the data found in the $ES(m)$ and $V_{reach}(es)$ maps.

SPAIN's initialization script loads the kernel module, configures SPAIN-specific MAC and IP addresses for the openflow virtual network interface, and starts the local controller process. The controller then contacts a central controller to obtain the necessary maps. We also exploit this central controller (repository) to provide weights for choosing between VLANs (to support traffic engineering) and a map showing which links are in each VLAN; the latter would be useful in conjunction with a link-monitoring service that informs an end-host when a link fails or recovers. In our current implementation, a host only communicates with the repository at initialization time

Our host controller communicates with the repository using XML RPCs, over a separate control network provided by our testbed. (The control network is not strictly necessary, but increases our ability to recover from failures that partition the data network.)

During normal operation, the OpenFlow module looks up each outgoing packet in its flow table, to find the VLAN tag to use for sending the packet. For the first packet of flow initiated from this host, the resulting lookup miss causes the user-mode controller to be invoked. The controller consults its tables to choose a VLAN; when multiple VLANs are available, the choice is random, biased by the weights provided by the central controller. The local controller inserts the appropriate rule in the flow table. If a new flow is initiated from another host, we record the incoming packet's VLAN tag, and use that VLAN for sending packets on that flow.

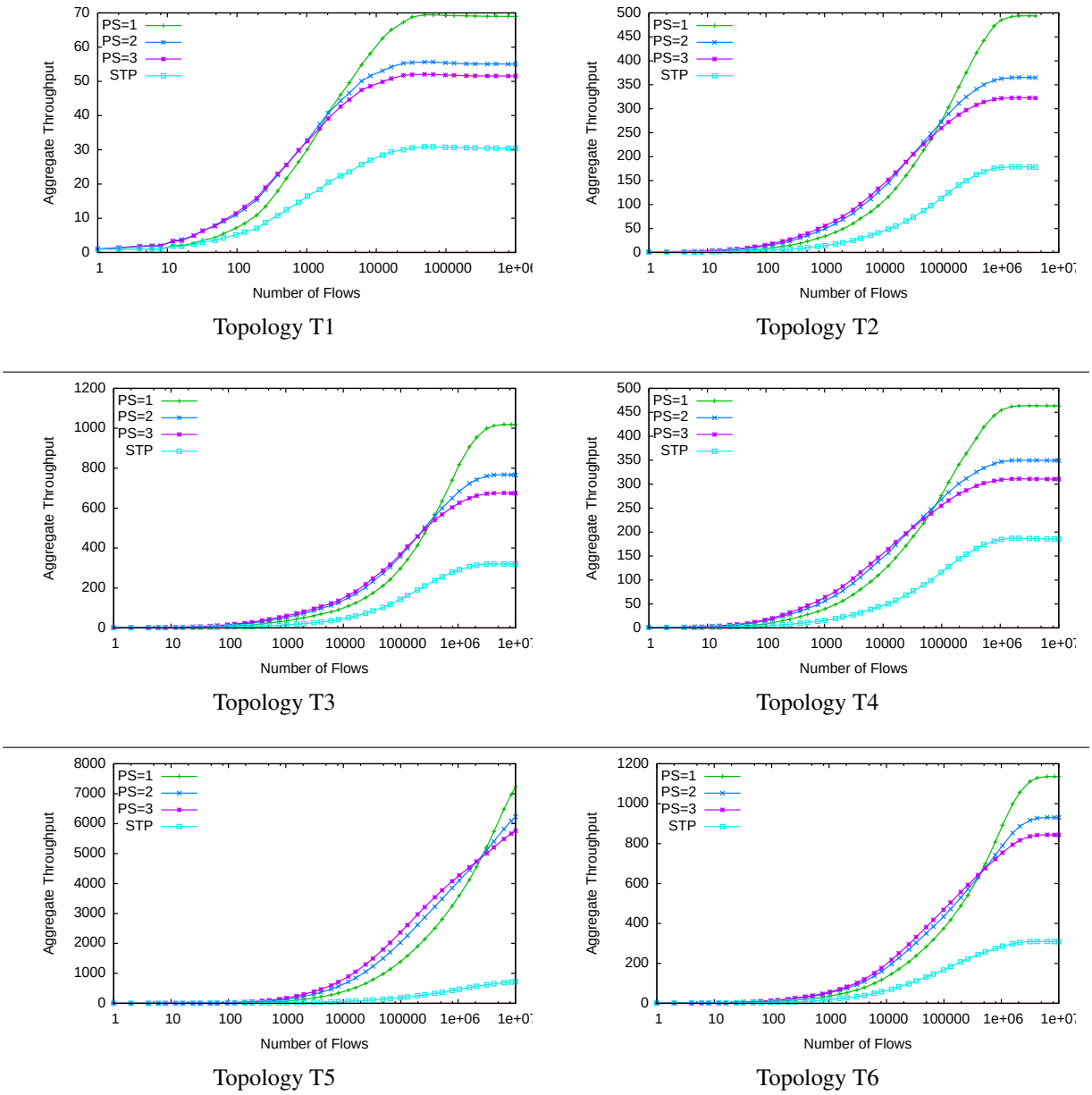


Figure 10: Aggregate throughput vs. number of flows: SPAIN and Spanning Tree on abitrary RocketFuel topologies.

ARP packets, and packets for non-SPAIN hosts, are sent without VLAN tags; the switch will forward these on the default VLAN.

Implementation glitches: While debugging our experimental evaluation, we realized that the dual-port NICs in our server machines were stripping the VLAN tags from incoming packets on one of their two ports.[5] This defeats several of the mechanisms that SPAIN's end-host algorithms, including the chirping protocol. (The same problem might arise when using a TCP Offload Engine, and might be harder to solve.) As a temporary workaround, we had to emulate the VLAN-based mechanisms of SPAIN by directly updating the per-host FIB entries in each switch. We have already mentioned such "FIB-pinning" (Sec. 5.7 as an alternate way to implement SPAIN's basic multipathing mechanism, but it potentially consumes more switch table resources.

[5]We believe we can update the NIC firmware to avoid this, but due to the need for per-server manual updates using a DOS-based diagnostic tool, have not yet done so on the 84 servers in our testbed.

Topo.	Throughput (aggregate data per unit time				Normalized w.r.t Spanning Tree		
	Spanning Tree	SPAIN			SPAIN		
		PS=1	PS=2	PS=3	PS=1	PS=2	PS=3
T1	30.45	68.97	55.04	51.54	2.26	1.81	1.69
T2	178.17	493.71	364.93	322.32	2.77	2.05	1.81
T3	318.77	1018.60	766.36	674.42	3.20	2.40	2.12
T4	185.45	463.56	349.42	310.55	2.50	1.88	1.67
T5	720.72	7232.69	6233.68	5762.97	10.04	8.65	8.00
T6	309.15	1136.33	930.85	843.54	3.68	3.01	2.73
	Spanning Tree	PS = $(p/2)^2$			PS = $(p/2)^2$		
FT4	8	16			2		
FT8	32	128			4		
FT16	128	1024			8		
FT48	1152	27648			24		

Table 5: Aggregate throughput for different topologies with up to 10 million flows (saturates all topologies except T5).

9.2 Approach 2: User-mode

We were also able to implement the basic functions of SPAIN without any kernel changes at all, using two Linux features available in all distributions: the 802.1Q module that adds VLAN support, and the bonding driver module, which "bonds" several interfaces into a single virtual-Ethernet interface.

```
# step 1: insert VLAN and bonding modules
    modprobe 8021q
    modprobe bonding
      # reads a SPAIN-specific config file

# step 2: create per-VLAN virtual interfaces
    for i in `cat spain.vlans`
    do
        vconfig add eth0 $i
    done

# step 3: bond all virtual interfaces into one
    ifconfig bond0 $MY_IPADDR netmask $MY_MASK
    for i in `cat spain.vlans`
    do
       ifenslave bond0 eth0.$i
    done

# step 4: adjust the routing table
    eval `netstat -rn | \
       awk -f spainRoutesAdjust.awk`
    # AWK script generates the necessary
    # "route add" commands
```

Figure 11: SPAIN in Linux without kernel changes

Figure 11 is an approximate version of the configuration script that implements SPAIN. The script, which must run as root, starts by installing the necessary modules.

The script then creates virtual interfaces for all VLANs, using a list of VLANs extracted from the reachability map that the host has previously downloaded from the repository. Each such device adds the appropriate VLAN tag to outgoing packets, and only receives packets with the appropriate VLAN tag. Although sec. 5.5 shows that we should not need many such virtual interfaces, we found that the VLAN module is relatively space-efficient, and adding the maximum 4096 virtual interfaces would consume only 5MB of RAM. We also believe that the per-packet overheads are independent of the number of VLANs configured.

The script then bonds all of the VLAN-specific virtual interfaces into a single `bond0` virtual interface.

Finally, we modify the routing table so that the routes for SPAIN end hosts point to the `bond0` interface.

Loading the `bonding` module causes it to read a configuration file, which we use to approximate the ideal SPAIN end-host behavior via these parameters:

- `mode`: set to `balancetlb` to load-balance outgoing flows across all of the bonded VLANs.
- `xmit_hash_policy`: specifies how flows are defined; we use `layer3+4` to define flows using the standard TCP/UDP 5-tuple. The module's hash function is based on `XOR`, which means that for many regular topologies, both directions of a flow are hashed to the same VLAN; this reduces the need for chirping and switch-learning floods.

Two more parameters, `arp_ip_target` and `arp_interval`, help further reduce flooding by allowing us to approximate a periodic chirp protocol. However, setting these parameters correctly is somewhat tricky, and we are still working on a robust method for choosing their values.

Our user-mode implementation has several limitations: (1) it assumes that all VLANs reach all SPAIN end hosts in the system, and (2) if a link on a VLAN goes down, to ensure connectivity a control function must remove, from all hosts, the virtual interface corresponding

to that VLAN. Note that on a link failure, an ideal SPAIN end-point implementation will still use that VLAN for other destinations that remain reachable on that VLAN.

10 Experimental evaluation

In our experiments, we evaluate four aspects of SPAIN: overheads added by the end-host software; how SPAIN improves over a traditional spanning tree; support for incremental deployment; and tolerance of network faults.

We do not compare SPAIN's performance against other proposed data-center network designs, such as PortLand [27] or VL2 [18], because these require specific network topologies. SPAIN's support for arbitrary topology is an advantage: one can evaluate it on the topology one has access to. (We plan to rebuild our testbed network to support fat-tree topologies, but this is hard to do at scale.)

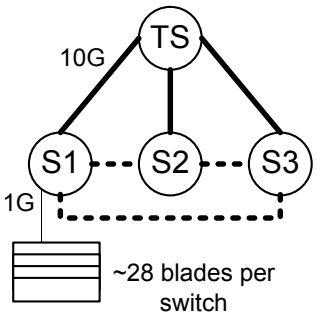
10.1 Configuration and workloads

We conducted our evaluation on three racks in the (larger) Open Cirrus testbed [12]. Each rack consisted of of about 28 machines each. These machines have quad-core 2GHz Intel Xeon CPUs, 8GB RAM and run Ubuntu 9.04. Due to some servers with failed disks, we were not able to use all $3 * 28 = 84$ servers in our experiments. Our experiments with the OpenFlow-based SPAIN implementation used 80 servers, and those with the user-mode SPAIN implementation used 82 servers.

Each server is connected to a 3500-series ProCurve switch using a 1-GigE link, and these rack switches are connected to a central 5406-series ProCurve switch via 10-GigE links.

The Open Cirrus cluster was originally wired using a traditional two-tiered tree, with the core 5406 switch (TS) connected to the 3500 switches (S1, S2, and S3) in each logical rack. To demonstrate benefits of SPAIN, we added 10-GigE cross-connects between the 3500 switches, so that each such switch is connected to another switch in each physical rack. Fig. 12 shows the resulting wired topology, and Fig. 13 shows the four VLANs computed by the SPAIN offline configuration algorithms.

In our tests, we used a "shuffle" workload (similar to that used in [18]), an all-to-all memory-to-memory bulk transfer among N participating hosts. This communication pattern is used in several important applications, such as the shuffle phase between Map and Reduce phases of MapReduce, and in join operations in large distributed databases. In our workload, each host transfers 500MB to every other host using 5 simultaneous threads; the order in which hosts choose destinations is randomized to avoid deterministic hot spots. With each machine sending 500MB to all other machines, this experiment transfers about 3.16TB to 3.32TB, depending on the number of machines used.



Dashed lines represent the non-spanning-tree links that we added.

Figure 12: Wiring topology used in our experiments.

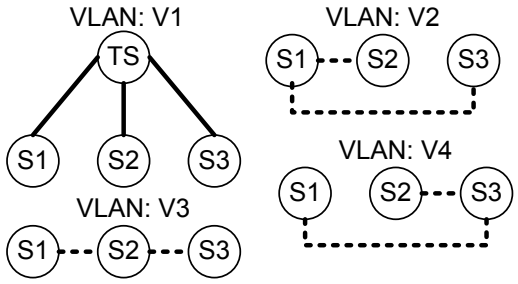


Figure 13: VLANs used by SPAIN for our topology

10.2 End-host overheads

We measured end-host overheads of several configurations, using ping (100 trials) to measure latency, and NetPerf (10 seconds, 50 trials) to measure both uni-directional and simultaneous bi-directional TCP throughput between a pair of hosts.

Table 6: End-host overheads

Configuration	ping RTT (usec)	TCP throughput (Mbit/sec)	
		1-way	2-way, [min,max]
Unmodified Linux	97	941	1463 [1108,1680]
1st pkt, cold start	1.1 ms		
SPAIN/OpenFlow	113	939	1298 [1231, 1361]
1st pkt, cold start	2-4ms		
SPAIN/user-mode	98	939	1435 [1196, 1719]
1st pkt, cold start	5-8ms		

ping results: mean of 100 warm-start trials;
throughput: mean, min, max of 50 trials

Table 6 shows the results. The OpenFlow-based implementation of SPAIN somewhat increases RTT (especially for the first packet of a flow, which invokes the user-level controller), but does not appreciably change one-way TCP performance, which is near optimal.

However, the bidirectional throughput, even for unmodified Linux is much less than twice the one-way throughput, and shows a high variance; we are investigating possible causes of this variance.

Using the OpenFlow-based SPAIN, we see approx-

imately 10% lower two-way throughput, and a doubling of CPU utilization, compared to unmodified Linux. However, our user-mode SPAIN implementation delivers TCP throughput essentially identical to that of unmodified Linux. This suggests that using the OpenFlow module for this purpose is unnecessarily inefficient; a custom SPAIN kernel module would yield better two-way throughput, and probably would almost eliminate the cold-start RTT penalty.

10.3 SPAIN vs. spanning tree

Tables 7 and 8 show how SPAIN compares to spanning tree when running the shuffle workload on the Open Cirrus testbed. SPAIN improves aggregate goodput by 28.5% with our user-mode implementation, and by 20% with the OpenFlow-based implementation. At least in the latter case, the improvement from SPAIN is clearly limited by CPU overheads rather than network utilization.

Table 7: Spanning-tree vs. OpenFlow-based SPAIN

	Spanning Tree	**SPAIN/OpenFlow**	
		100%	**80%**
Mean goodput/host (Mb/s)	480.87	576.67	599.63
Aggregate goodput (Gb/s)	38.41	46	47.76
Mean completion time/host	690 s	578 s	554 s
Total shuffle time	717 s	634 s	599 s

Results are means of 5 trials, 500 MBytes/trial, 80 hosts

Table 8: Spanning-tree vs. user-mode SPAIN

	Spanning Tree	**SPAIN/u-mode**
Mean goodput/host (Mb/s)	469.30	604.80
Aggregate goodput (Gb/s)	38.42	49.38
Mean completion time/host	724.92 s	564.38 s
Total shuffle time	768.47 s	617.66 s

Results are means of 10 trials, 500 MBytes/trial, 82 hosts

When we used the OpenFlow-based implementation, we measured an aggregate goodput of 46 Gbps, which is only 57.5% of the ideal goodput (80 Gbps) one would expect. We believe this discrepancy is caused by several limits:

1. The inability of our individual hosts to saturate their 1 Gbps links with bidirectional TCP traffic (see Table 6). Even with unmodified Linux, for bidirectional TCP tranfers we observe only 71% of the ideal throughput ($1463/2000 = 0.71$). This would imply, for our 80-node shuffle experiment using spanning tree, an upper bound of $80 * 0.71 = 56.8$ Gbps. With the OpenFlow-based implementation of SPAIN, for a single bidirectional TCP transfer we observed 65% of the ideal goodput ($1298/2000 = 0.65$), which implies a maximum aggregate goodput, for the 80-node shuffle, of $80 * 0.65 = 52$ Gbps.

2. The OpenFlow kernel module adds another overhead, not visible in the single-connection tests of Table 6. In our implementation, we install a wildcard OpenFlow rule for each open connection to a SPAIN peer. However, we later realized that the OpenFlow module uses a linear search of the wildcard rule table. The shuffle experiment leads to the installation of a large number of wild-card rules (especially since they persist after TCP connections are closed), and this appears to further reduces the measured aggregate goodput by 7.5%.

   Table 7 shows a column for a configuration with 80% (64) of the nodes running SPAIN, and 20% using spanning tree. Note that these results are better than either of the other two columns; this supports our belief that the linear search limits performance in the shuffle experiment.

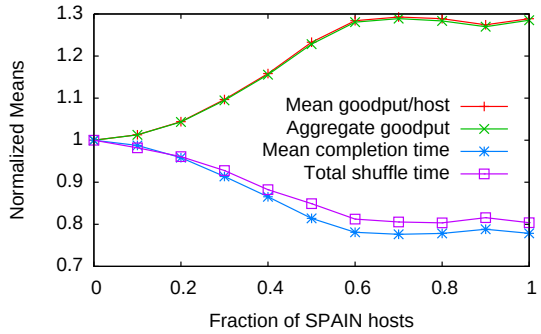
Our experiments with the user-mode implementation of SPAIN, which should avoid these two problems, yielded an aggregate goodput of 49.38 Gbps, which is 60.22% of the ideal 82-node goodput of 82 Gbps. This is about 10% less than one would expect based on the two-node bidirectional TCP transfer measurement shown in Table 6 ($1435/2000 = 0.705$ or 70.5%). We are not yet sure what causes this discrepancy.

10.4 Incremental deployability

One of the key features of SPAIN is incremental deployability. To demonstrate this, we randomly assigned a fraction f of hosts as SPAIN nodes, and disabled SPAIN on the remaining hosts. We measured the goodput achieved with the shuffle experiment. To account for variations in node placement, we ran N trials for each fraction f, doing different random node assignments for each trial We used $N = 5$ for experiments with the OpenFlow-based implementation, and $N = 10$ for experiments with the user-mode implementation.

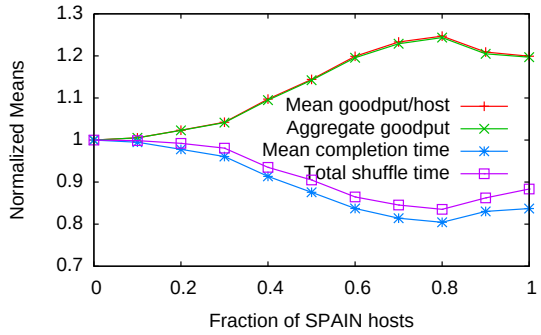
In Fig. 14, with the user-mode implementation of SPAIN, we show how several metrics (per-host goodput, aggregate goodput, mean per-host completion time, and total shuffle run-time) vary as we change the fraction f of SPAIN vs. non-SPAIN nodes. The y-values for each curve are normalized to the 0%-SPAIN results. The curves flatten out at about $f = 0.6$, probably for the same reason(s) that we see about 10% less aggregate goodput than we expect for the $f = 1.0$ case.

Fig. 15 shows similar measurements made with the OpenFlow-based implementation of SPAIN. Note that the best-case performance is worse than that for the user-mode implementation. Note also that the results for $f \geq 0.9$ are depressed because of another overhead with



Results are means over 10 trials

Figure 14: Incremental deployability (user-mode SPAIN)



Results are means over 5 trials

Figure 15: Incremental deployability (OpenFlow-based SPAIN)

the OpenFlow module – we started getting misses in its 100-entry wildcard flow cache.

10.5 Fault tolerance

We implemented a simple fault detection and repair module that runs at user-level, periodically (100 msec) monitoring the performance of flows. It detects that a VLAN has failed for a destination if the throughput drops by more than 87.5% (equivalent to three halvings of the congestion window), in which case it re-pins the flow to an alternate VLAN.

To demonstrate fault-tolerance in SPAIN, we ran a simple experiment. We used NetPerf to generate a 50-second TCP flow, and measured its throughput every 100 msec. Fig. 16 shows a partial time-line. At 29.3 sec., we removed a link that was in use by this connection. SPAIN detects the failure and repairs the end-to-end path; the TCP throughput returns to normal within 200–300 msec.

11 SPAIN and the broadcast vs. scaling problem

SPAIN does not eliminate the problem of broadcasts, but we can exploit certain aspects of both the data-center environment and our willingness to modify end-host implementations.

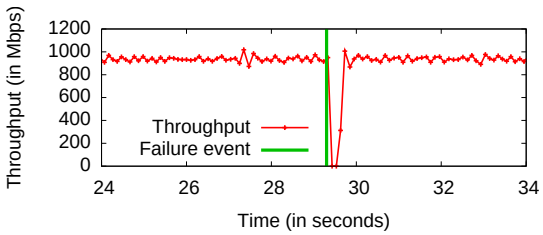


Figure 16: Fault-tolerance experiment

The most common broadcast packets on an Ethernet are ARP packets. There are other source of broadcasts such as DHCP, NetBIOS, and Internet Printing Protocol, but they are not common in data center networks and are typically very small in volume compared to the data traffic. Hence, focus here is only on ARP-related packets.

11.1 Review of ARP

The Address Resolution Protocol (ARP) is used by IP hosts to determine the MAC address of another host whose IP address is known. ARP relies on the broadcasting capability of the underlying Ethernet [31]. An end host X broadcasts an ARP request packet that includes the IP address that it is trying to resolve. Each end host Y that receives an ARP request compares the IP address in the ARP message against its own IP address. If it matches, Y creates an ARP reply packet including both the requested IP address and its MAC address, then sends the reply packet to X as a unicast Ethernet packet.

Each end host caches, in a local ARP table, the IP→MAC address bindings that it has resolved. It also maintains a timestamp along with each binding denoting the latest time at which it received a packet that informed this binding. These bindings, if they were to persist forever, could become stale – that is, fail to reflect reality, due to host mobility, reassignment of an IP address to a different host, or some fault-tolerance methods. Therefore, any ARP-table entry with a time stamp older than *arp-timeout* is treated as stale and must be refreshed. Most Linux Kernels use a value of 30 seconds for this timeout [1, 13]. If a packet needs to be sent to a destination IP with stale ARP-table entry, the ARP specification [31] suggests (and many current operating systems implement) sending a unicast ARP request, since a possible MAC address is already known. If that fails to elicit a response, then the stale entry is removed and the ARP request is re-transmitted as a broadcast.

An existing ARP-table timestamp for the entry for IP address A is reset when the local host receives an ARP request from A. This also suppresses the need to send a broadcast ARP request in some cases – for example, when host A initiates an IP flow to host B, A will first broadcast an ARP request for B, but B will not need to broadcast a request for A.

Therefore, ARP requests are not sent as broadcasts as

long as there is an entry in the ARP table for the corresponding IP address.

11.2 Chirping can suppress ARP broadcasts

SPAIN's chirping protocol (see Sec. 6) can suppress additional ARP broadcasts by functioning analogously to the Gratuitous ARP (GARP) mechanism, in which a host broadcasts an ARP request for its own IP → MAC binding. All hosts that receive this broadcast will update their ARP-table entries for the IP address, and reset the corresponding timestamps.

GARP was first designed to detect IP address conflicts [40]. It has also been used in supporting IP mobility [28], and as the underlying mechanism for *Virtual IP*, a scheme for load balancing and failing over across server groups [13]. VM migration exploits this existing mechanism to proactively update the forwarding table entries in the switches, because the switches learn from the GARP messages as if it were a regular ARP message.

Chirping functions analogously to GARP, but without the need to send broadcasts; as shown in Algorithm 8, incoming (unicast) chirp packets carry sufficient information to update the local ARP table entry for the remote host.

11.3 Other techniques for reducing ARP broadcasts

Although not part of SPAIN's design *per se*, there are other techniques that can reduce the ARP broadcast rate in a data center.

For example, one can increase the ARP-table (cache) size on server hosts to be large enough to hold the entries for *all* other hosts in the network. For a 100,000-node network this requires just a few Megabytes of additional memory [13]; this is less than 1% of the 4-32 Gigabytes of memory provisioned in typical data center machines.

12 Summary and conclusions

Our goal for SPAIN was to provide multipath forwarding using inexpensive, COTS Ethernet switches, over arbitrary topologies, and support incremental deployment. We have demonstrated, both in simulations and in experiments, that SPAIN meets those goals. In particular, SPAIN improves aggregate goodput over spanning-tree by 28.5% on a testbed topology that would not support most other scalable-Ethernet designs.

References

[1] ARP Cache Implementation in Linux Kernel 2.6.28.8. The exact line of source code shown at: http://lxr.linux.no/linux+v2.6.28.8/net/ipv4/arp.c#L184.

[2] ARP Flooding Attack. http://www.trendmicro.com/vinfo/secadvisories/default6.asp?VNAME=ARP+Fl%ooding+Attack.

[3] Campus network for high availability: Design guide. Cisco Systems, http://tinyurl.com/d3e6dj.

[4] Enterprise campus 3.0 architecture: Overview and framework. Cisco Systems, http://tinyurl.com/4bwr33.

[5] FocalPoint in Large-Scale Clos Switches. White Paper, Fulcrum Microsystems. http://www.fulcrummicro.com/documents/applications/clos.pdf.

[6] IEEE P802.3ba 40Gb/s and 100Gb/s Ethernet Task Force. http://www.ieee802.org/3/ba/public/index.html.

[7] IETF TRILL Working Group. http://www.ietf.org/html.charters/trill-charter.html.

[8] Woven Systems unveils 10G Ethernet switch. Network World, http://tinyurl.com/ajtr4b.

[9] 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.

[10] M. Al-Fares, A. Loukissas, and A. Vahdat. A scalable, commodity data center network architecture. In *Proc. SIGCOMM*, pages 63–74, 2008.

[11] M. Arregoces and M. Portolani. *Data Center Fundamentals*. Cisco Press, 2003.

[12] R. Campbell et al. Open Cirrus Cloud Computing Testbed: Federated Data Centers for Open Source Systems and Services Research. In *Proc. HotCloud*, 2009.

[13] Christian Benvenuti. *Understanding Linux Network Internals*. O'Rielly Media Inc., Dec. 2005.

[14] C. Clos. A Study of Non-Blocking Switching Networks. *Bell System Technical Journal*, 32(2):406–424, 1953.

[15] K. Elmeleegy and A. L. Cox. EtherProxy: Scaling Ethernet By Suppressing Broadcast Traffic. In *Proc. INFOCOM*, 2009.

[16] A. Gonsalves. Interop: Broadcom Says 40 Gbps Ethernet Coming Soon. Information Week, http://tinyurl.com/cakarl.

[17] A. Greenberg, J. Hamilton, D. A. Maltz, and P. Patel. The Cost of a Cloud: Research Problems in Data Center Networks. *SIGCOMM CCR*, 2009.

[18] A. Greenberg, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta. VL2: A Scalable and Flexible Data Center Network. In *Proc. SIGCOMM*, Barcelona, 2009.

[19] A. Greenberg, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta. Towards a Next Generation Data Center Architecture: Scalability and Commoditization. In *Proc. PRESTO*, pages 57–62, 2008.

[20] 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.

[21] C. Guo, H. Wu, K. Tan, L. Shiy, Y. Zhang, and S. Luz. DCell: A Scalable and Fault-Tolerant Network Structure for Data Centers. In *Proc. SIGCOMM*, Aug. 2008.

[22] C. Kim, M. Caesar, and J. Rexford. Floodless in SEATTLE: A Scalable Ethernet Architecture for Large Enterprises. In *Proc. SIGCOMM*, pages 3–14, 2008.

[23] M. Ko, D. Eisenhauer, and R. Recio. A Case for Convergence Enhanced Ethernet: Requirements and Applications. In *Proc. IEEE ICC*, 2008.

[24] K.-S. Lui, W. C. Lee, and K. Nahrstedt. STAR: a transparent spanning tree bridge protocol with alternate routing. *SIGCOMM CCR*, 32(3), 2002.

[25] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. OpenFlow: enabling innovation in campus networks. *SIGCOMM CCR*, 38(2):69–74, 2008.

[26] A. Myers, T. S. E. Ng, and H. Zhang. Rethinking the Service Model: Scaling Ethernet to a Million Nodes. In *Proc. HOTNETS-III*, 2004.

[27] 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.

[28] C. Perkins. IP Mobility Support. RFC 2002 Available at: `http://www.ietf.org/rfc/rfc2002.txt`.

[29] R. Perlman. An Algorithm for Distributed Computation of a Spanning Tree in an Extended LAN. *SIGCOMM CCR*, 15(4), 1985.

[30] R. J. Perlman. Rbridges: Transparent Routing. In *Proc. INFOCOM*, 2004.

[31] D. C. Plummer. An Ethernet Address Resolution Protocol. RFC 826 Available at: `http://www.ietf.org/rfc/rfc826.txt`.

[32] T. L. Rodeheffer, C. A. Thekkath, and D. C. Anderson. SmartBridge: A Scalable Bridge Architecture. In *Proc. SIGCOMM*, 2000.

[33] Sam Halabi. *Metro Ethernet*. Cisco Press, 2003.

[34] M. Scott, A. Moore, and J. Crowcroft. Addressing the Scalability of Ethernet with MOOSE. In *Proc. DC CAVES Workshop*, Sept. 2009.

[35] R. Seifert and J. Edwards. *The All-New Switch Book: The Complete Guide to LAN Switching Technology*. Wiley, 2008.

[36] S. Sharma, K. Gopalan, S. Nanda, and T. Chiueh. Viking: a Multi-spanning-tree Ethernet Architecture for Metropolitan Area and Cluster Networks. In *Proc. INFOCOM*, 2004.

[37] N. Spring, R. Mahajan, and D. Wetherall. Measuring ISP Topologies with Rocketfuel. In *Proc. SIGCOMM*, 2002.

[38] A. Tavakoli, M. Casado, T. Koponen, and S. Shenker. Applying NOX to the Datacenter. In *Proc. HotNets*, 2009.

[39] Tommy, R, Jensen and Bjarne Toft. *Graph Coloring Problems*. Wiley InterScience, 1992.

[40] W. Richard Stevens. *TCP/IP Illustrated Volume 1: The Protocols*. Addison Wesley, Reading, MA, USA., 1994.