SPAIN: COTS Data-Center Ethernet for Multipathing over Arbitrary Topologies

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) Ethernet 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 experimental 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 [28] – 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 guarantee specific QoS properties.

In SPAIN, an offline network controller system first pre-computes a set of paths that exploit the redundancy in a given network topology, 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. The controller then merges these paths into a set of trees and maps each tree onto a separate VLAN, exploiting the VLAN support in COTS Ethernet switches. In most cases, only a small number of VLANs suffice 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 limited; in Linux, they consist of a loadable kernel module and a user-level controller.

We have evaluated SPAIN both in simulation and in an experimental deployment on a network testbed. We show that SPAIN adds virtually no end-host overheads, that it significantly improves 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 [18], the 40Gbps standard is expected this year [21], 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 [2, 3, 10].

2.1 Why we want Ethernet to scale

The use of multiple IP subnets, especially within a data center, creates significant management complexity. In particular, it requires a network administrator to simultaneously and consistently manage the L2 and L3 layers, even though these are based on very different forwarding, control, and administrative mechanisms.

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 [2, 3], 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) [28] 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-destination 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 designated (high-bandwidth) root switch 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 switch 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-destination pair, the higher the common ancestor in the spanning tree, the higher the number of competing source-destination 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-destination 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, 18].

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 [32]. 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 [1, 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 an Ethernet domain [3, 15, 22, 26]. Greenberg *et al.* [16] 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 [25] for more discussion.

3 Related Work

Spanning trees in Ethernet have a long history. The original algorithm was first proposed in 1985 [28], 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 [33]. 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.* [30], 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 inter-operate with existing bridges, and can also be optimized for IP [29]. Currently, this work is being standardized by the TRILL working group of IETF [7]. Note that TRILL focuses only on shortest-path or equal-cost routes, and does not support multiple paths of different lengths.

Myers *et al.* [26] proposed eliminating 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] proposed 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

Table 1: Comparing SPAIN against related work

	SPAIN	SEATTLE [22]	TRILL [7]	VL2 [17]	PortLand [27]	MOOSE [31]
Wiring Topology	Arbitrary	Arbitrary	Arbitrary	Fat-tree	Fat-tree	Arbitrary
Usable paths	Arb. multiple paths	Single Path	ECMP	ECMP	ECMP	Single Path
Deploy incrementally?	YES	NO	YES	NO	NO	YES
Uses COTS switches?	YES (L2)	NO	NO	YES (L3)	NO	NO
Needs end-host mods?	YES	NO	NO	YES	NO	NO

Fat-tree = multi-rooted tree; ECMP = Equal Cost Multi-Path.

of some of the prior approaches; otherwise, SEATTLE is quite similar to TRILL.

Greenberg *et al.* [18] proposed 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.* [31] similarly use MAC-address re-writing in MOOSE, but without imposing a specific topology; however, MOOSE uses shortest-path forwarding, rather than multipath.

VL2 [17] 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 achieves Equal-Cost Multipath (ECMP) forwarding in Clos topologies by assigning a single IP anycast address to all core switches. It is not obvious how one could assign such IP anycast addresses to make multipath forwarding work in non-Clos 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 [4]. 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 [20] and BCube [19].

As summarized in Tab. 1, SPAIN differs from all of this prior work because it provides multipath forwarding, 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 Fibre Channel over Ethernet (FCoE), that are required for "fabric convergence" within data centers [23]. Fabric convergence, the replacement of special-purpose interconnects such as Fibre Channel 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.

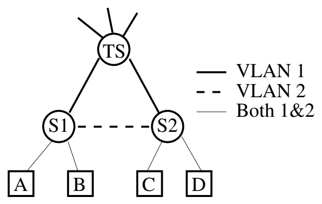


Figure 1: Example of VLANs used for multipathing

4.1 Overview of SPAIN

In SPAIN, we pre-compute a set of paths that utilizes the redundancy in the physical wiring, both to provide high bisection bandwidth and to improve fault tolerance. We then merge these paths into a set of trees, 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 ID can be re-used for 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 switch features: MAC-address learning and VLAN support; these are already present in most COTS switches. 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 the Per-VLAN Spanning Tree Protocol or the Multiple Spanning Tree Protocol (see Sec. 5.5).

SPAIN requires a switch to store multiple table entries (one per VLAN tree) for each destination, in the worst case where flows are active for all possible (VLAN, destination) pairs. (Table overflows lead to packet flooding; they are not fatal.) This could limit SPAIN's applicability to very large networks with densely populated traffic matrices, but even inexpensive merchant-silicon switches have sufficiently large tables for moderately-large networks.

For data centers where MAC addresses are known a priori, we have designed another approach called *FIB-pinning*, but do not describe it here due to space constraints. See [25] for more details.

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 (or equal-cost multi-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 utilize 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.

Because of space constraints, we omit many details of the path computation algorithms; these may be found in the Technical Report version of the paper [25].

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.

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 a limited number of paths.

[1] This is the terminology used by HP ProCurve. Cisco uses the terms *access mode* 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:  Initialize: $\forall e \in E : w(e) = 1$
8:  /* shortest computes weighted shortest path */
9:  Path $p$ = shortest$(G, s, d, w)$ ;
10: for $e \in p$ do
11:    $w(e) += |E|$
12:
13: while $(|P| < k)$ do
14:    $p$ = shortest$(G, s, d, w)$
15:    if $p \in P$ then
16:       /* no more useful paths */
17:       break ;
18:    $P = P \bigcup \{p\}$
19:    for $e \in p$ do
20:       $w(e) += |E|$
21:
22: return $P$
```

We accomplish this in steps shown in Algorithm 1. (This algorithm has been simplified to assume unit edge capacities; the extension to non-uniform weights is straightforward.)

First, (lines 7–11), we initialize the set of paths for each source-destination pair to include the shortest path. Shortest paths are attractive because in general, they minimize the network resources needed for each packet, and 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.

Then (lines 13–20), 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. As shown in [25], 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 10–11 and 19–20, 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.

5.3 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 Ethernet switches support 4096 VLANs, sometimes fewer.)

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.)

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.*

We prove in [25] that Problem 1 is NP-hard. Therefore, 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 6–12). If the greedy packing step fails for a path, a new graph is created with this path, and is added to SG (lines 13–15).

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.

The serial nature of Algorithm 2 does not scale well; its complexity is $O(mkn^3)$, where m is the VLANs, k is the number of paths, and n is the number of switches. We have designed a parallel algorithm, described in [25], based on graph-coloring heuristics, which yields speedup linear in the number of edge switches.

Algorithm 2 Greedy VLAN Packing Heuristic

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

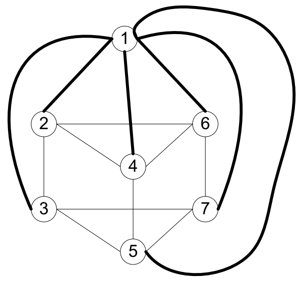


Figure 2: 7-switch topology; original tree in bold

5.3.1 An example

Fig. 2 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.

Fig. 3 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 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. (See [25] for more details about this analysis.)

These topologies include FatTree (p) [10], a 2-ary 3-tree, where p is the number of ports per switch; BCube (p, l) [19], where p is the number of ports per switch, and l is the number of levels in the recursive construction of

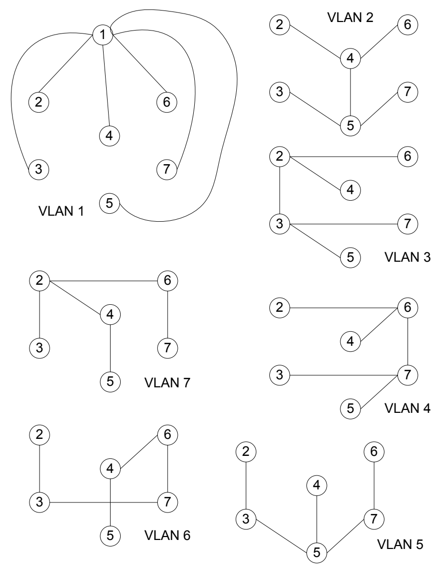


Figure 3: VLANs covering 7-switch topology of Fig. 2

the topology; 2-D HyperX (k) [9], where k is the number of switches in each dimension of a 2-D mesh; and CiscoDC, Cisco's recommended data center network [12], a three-layer tree with two core switches, and with parameters (m, a) where m is the number of aggregation modules, and a the number of access switch pairs associated with each aggregation module.

Table 2: 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

Table 2 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 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).

5.5 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.

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. We generically refer to the end-host implementation as the "SPAIN driver," although (as described in Sec. 9), some functions run in a user-mode daemon rather than in the kernel-mode 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.

[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.

- $V_{reach}(es)$: 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 call *chirping* for several functions: to avoid most timeout-related flooding, to test VLANs for usability, and to support virtual-machine migration. For VM migration, chirping works analogously to the Gratuitous ARP (GARP) mechanism, in which a host broadcasts an ARP request for its own IP $\rightarrow$ MAC binding.

An end-host A sends a unicast *chirp* packet to another host B if B has just started sending a flow to A, and if A has not sent any packets (including chirps) in the recent past (T_{sent}) to any host connected to the same edge switch as B. An end-host (possibly a virtual machine) broadcasts a chirp packet when it reboots and when it moves to a different switch. A chirp packet carries the triple <IP Address, MAC address, Edge-switch ID>. Chirp packets also carry a $want\_reply$ flag to trigger a unicast chirp in response; broadcast chirps 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, a host Y that fails to receive the broadcast chirp from host X 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.

Algorithm 3 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: /* see if any of the candidates are usable */
10: $usable\_vlans = candidate\_vlans \bigcap V_{usable}(es)$
11: if $usable\_vlans$ is empty then
12:     return the default VLAN (VLAN 1)
13: init_probe($candidate\_vlans - usable\_vlans$)
14: return a random $v \in usable\_vlans$.
```

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 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 3 shows the procedure for VLAN selection for a flow to a destination MAC m. The driver uses the $ES(m)$ to find the edge switch es and then uses the reachability map R to find the set of VLANs that reach es. If m does not appear in the ES table, then the driver uses the default VLAN for this flow, and sends a unicast chirp to m to determine if it is a SPAIN host. 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 5).

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)

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

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 usable 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. Re-pinning 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 the SPAIN driver 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. The SPAIN driver 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 4 illustrates the decision process for re-pinning a flow; it is invoked whenever the flow attempts to send a packet.

One risk of re-pinning based on decreases in the congestion window is that it could lead to instability if many flows are sharing a link that suddenly becomes overloaded. SPAIN tries to prevent oscillations by spreading out the re-pinning operations. Also, pinning a flow to a new VLAN does not cause the original VLAN to be marked as unusable, so new flow arrivals could still be assigned to that VLAN, which should damp oscillations. However, we lack solid evidence that these techniques guarantee stability; resolving this issue is future work.

Algorithm 4 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:  $current\_es$ = get_es($dst\_mac$)
5:  if $saved\_es$ (from the flow state) $! = current\_es$ then
6:      /* destination moved – update flow state & re-pin */
7:      $saved\_es = current\_es$;
8:      return true
9:  if current_vlan($flow$) $\leq 0$ then
10:     return true /* new flows need VLAN selection */
11: if proto_of($flow$) $\neq TCP$ then
12:     if $(now - last\_pin\_time) \geq T_{repin}$ then
13:         return true /* periodic re-pin */
14: else
15:     if cwnd($flow$) $\leq W_{repin\_thresh}$ && is_rxmt($flow$) then
16:         return true /* TCP flow might prefer another path */
17: return false /* no need to repin */
```

Algorithm 5 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($m, vlan$)
8:  $es$ = get_es($m$) /* determine sender's edge switch */
9:  /* mark packet-arrival VLAN as $usable$ for $es$ */
10: $V_{usable}(es) = V_{usable}(es) \bigcup vlan$
11: /* chirp if we haven't sent to $es$ via $vlan$ recently */
12: if the $vlan$ bit in $V_{sent}(es)$ is not set then
13:     send_unicast_chirp($m, vlan$)
14:     /* $V_{sent}(es)$ is cleared every $T_{sent}$ sec. */
15: deliver packet to protocol stack
```

6.3 Receiving a Packet

Algorithm 5 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.

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 3.

Finally, before delivering the received packet to the protocol stack, the SPAIN driver sends a unicast chirp

[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.

to the source host if one has not been sent recently. (The pseudo-code omits a few details, including the case where the mapping $ES(m)$ 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, our driver performs 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. In SPAIN, a VLAN is used to represent a collection of paths. Most failures only affect a subset of those paths. Hence, the usability of a VLAN to reach a given destination is a function of the location of the sender. When the sender moves, it has to re-learn this usability, so it flushes its usability map $V_{usable}(es)$.

Also, peer end-hosts and Ethernet switches must learn where the host is now connected. Therefore, after a host has finished its migration, it broadcasts a chirp, which causes the recipient hosts to update their ARP and ES tables, and which causes Ethernet switches to update their learning tables.

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 3).

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. The SPAIN driver 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, the SPAIN driver re-probes less aggressively, to avoid unnecessary network overhead.

7 How SPAIN meets its goals

We can now summarize 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: The correctness of SPAIN's end-host processing does not depend 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 a variety of regular topologies, as de-

Table 3: Summary of simulation results

				Coverage		NCP			Throughput gain	
Topology	#Switches	#Links	#Hosts	STP	SPAIN	STP	SPAIN	#VLANs	$PS = 1$	$PS = \alpha$
FatTree(4)	20	32	16	37.50	100.00	35.00	0.00	4	1.00	2.00
FatTree(8)	80	256	128	15.62	100.00	17.00	0.00	16	1.00	4.00
FatTree(16)	320	2048	1024	7.03	100.00	21.00	0.00	64	1.00	8.00
FatTree(48)	2880	55296	27648	2.17	100.00	17.00	0.00	576	1.00	24.00
HyperX(3)	9	18	216	44.44	100.00	10.83	0.00	12	3.02	1.81
HyperX(4)	16	48	384	31.25	100.00	21.41	0.00	38	4.38	2.49
HyperX(8)	64	448	1536	14.06	100.00	16.31	0.00	290	9.46	5.18
HyperX(16)	256	3840	6144	6.64	100.00	17.86	0.00	971	19.37	10.49
CiscoDC(2,2)	14	31	192	32.26	90.32	12.14	0.71	9	2.20	2.00
CiscoDC(3,2)	20	46	288	34.15	92.68	12.88	0.00	12	2.22	2.00
CiscoDC(4,3)	34	81	576	35.53	94.74	14.64	0.30	18	2.23	2.00
CiscoDC(8,8)	146	361	3072	37.67	97.51	17.91	0.23	38	2.24	2.00
BCube(8,2)	16	128	64	56.25	100.00	23.81	0.14	16	1.44	1.17
BCube(48,2)	96	4608	2304	51.04	100.00	22.50	0.36	96	1.04	1.04
BCube(8,4)	2048	16384	4096	31.82	100.00	43.19	0.00	2048	1.68	1.59

Key: *Coverage*= % of links covered by STP; *NCP*= % of node pairs with no connectivity, for link-failure probability = 0.04; *VLANs*= # VLANs required; *Throughput gain*= aggregate throughput, normalized to STP, for sufficient flows to saturate the network. *PS*= Path-set size; α = Maximum number of edge-disjoint paths between any two switches; $(p/2)^2$ for FatTree(p) topologies, $2k-2$ for HyperX, 3 for CiscoDC, and l for BCube.

fined in Section 5.4: FatTree (p), BCube (p, l), 2-D HyperX (k), and CiscoDC (m, a).

Table 3 summarizes some of our simulation results for these topologies. We show results where SPAIN's path-set size PS (the number of available paths per source-destination pair) is set to the maximum number α of edge-disjoint paths possible in each topology. $\alpha = (p/2)^2$ for the FatTree topologies, $\alpha = 2k - 2$ for HyperX, $\alpha = 3$ for CiscoDC, and $\alpha = l$ for BCube (l is the number of levels in a BCube(p, l) topology). For throughput experiments, we also present results for $PS = 1$.

The *Coverage* column shows the fraction of links covered by a spanning tree and SPAIN. Except for the CiscoDC topologies, SPAIN always covers 100% of the links. In case of the CiscoDC topologies, our computed edge-disjoint paths do not utilize links between the pairs of aggregation switches, nor the link between the two core switches. Hence, SPAIN's VLANs do not cover these links.

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 set to the maximum edge-disjoint paths; 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. For all topologies considered, the number is below Ethernet's 4K limit.

The *Throughput gain* columns show the aggregate throughput achieved through the network, normalized so that STP = 1. We assume unit per-link capacity and fair sharing of links between flows. We also assume that SPAIN chooses at random from the α available paths (for the $PS = \alpha$ column), and we report the mean of 10 randomized trials.

Our throughput simulation is very simple: it starts by queueing all N flows (e.g., 1 million) spread across H hosts, and then measures the time until all have completed. This models a pessimistic case where all host-to-host paths are fully loaded. Real-world data-center networks never operate like this; the experiments in Sec. 10 reflect a case in which a only subset of host-to-host paths are fully loaded. For example, SPAIN's throughput gain over STP for our BCube(48,2) simulations peaks at about 10x when the number of flows is approximately the number of hosts (this case is not shown in the table). Also, the simulations for SPAIN sometimes favor the $PS = 1$ configuration, which avoids the congestion that is caused by loading too many paths at once (as with $PS = \alpha$ case).

In summary, SPAIN's paths cover more than twice the links, and with significantly more reliability, than spanning-tree's paths, and, for many topologies and workloads, SPAIN significantly improves throughput over spanning tree.

9 Linux end-host implementation

Our end-host implementation for Linux involves two components: a dynamically-loadable kernel-module ("driver") that implements all data-plane functionality, and a user-level controller, mostly composed of shell and Perl scripts.

On boot-up (or whenever SPAIN functionality needs to be initialized) the user-level controller first determines the MAC address of the network interface. It also determines the ID of the edge-switch to which the NIC is connected, by listening to the LLDP messages sent by the switch. It then contacts a central repository, via a pre-configured IP address; currently, we hard-code the IP address of the repository, but it could be supplied to each host via DHCP options. The controller then downloads the reachability map V_{reach}, and optionally a table that provides bias weights for choosing between VLANs (to support traffic engineering).

The controller then loads the SPAIN kernel driver, creating a `spain` virtual Ethernet device. Next, the controller configures the `spain` virtual device, using the following three major steps.

First, the controller attaches the `spain` device, as a master, to the underlying real `eth` device (as a slave). This master-slave relationship causes all packets that arrive over the `eth` device to be diverted to the `spain` device. That allows SPAIN's chirping protocol to see all incoming packets before they are processed by higher-layer handlers.

Second, the controller configures the `spain` device with the same IP and MAC addresses as the underlying `eth` device. The controller adjusts the routing table so that all the entries that were pointing to the `eth` device are now pointing to `spain`. This re-routing allows SPAIN's chirping protocol to see all outgoing packets.

Third, the controller supplies the driver with the maps it downloaded from the central repository, via a set of special `/proc` files exposed by the driver.

The `spain` driver straightforwardly implements the algorithms described in Section 6, while accounting for certain idiosyncrasies of the underlying NIC hardware. For instance, with NICs that do not support hardware acceleration for VLAN tagging on transmitted packets, the driver must assemble the VLAN header, insert it between the existing Ethernet header fields according to the 802.1Q specification, and then appropriately update the packet meta-data to allow the NIC to correctly compute the CRC field. Similarly, NICs with hardware acceleration for VLAN reception may sometimes deliver a received packet directly to the layer-3 protocol handlers, bypassing the normal driver processing. For these NICs, the `spain` driver must install an explicit packet handler to intercept incoming packets. (Much of this code is borrowed directly from the existing 802.1q module.)

Data structures: The SPAIN driver maintains several tables to support VLAN selection and chirping. To save space, we only discuss a few details. First, we maintain multiple bitmaps (for V_{usable} and V_{sent}) representing several time windows, rather than one bitmap; this spreads out events, such as chirping, to avoid large bursts of activity. Our current implementation ages out the stored history after about 20 seconds, which is fast enough to avoid FIB timeouts in the switches, without adding too much chirping overhead.

We avoid the use of explicit timers (by letting packet events drive the timing, as in "soft timers" [11]), and the use of multiprocessor locks, since inconsistent updates to these bitmaps do not create incorrect behavior.

Overall, the driver maintains about 4KB of state for each known edge switch, which is reasonable even for fairly large networks.

Limitations: Our current implementation can only handle one NIC per server. Data-center servers typically support between two and four NICs, mostly to provide fault tolerance. We should be able to borrow techniques from the existing `bonding` driver to support simultaneous use of multiple NICs. Also, the current implementation does not correctly handle NICs that support TCP offload, since this feature is specifically intended to hide layer-2 packets from the host software.

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 and shortest-path routing; support for incremental deployment; and tolerance of network faults.

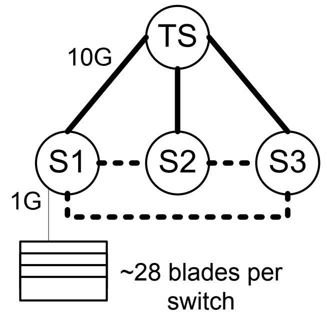
We do not compare SPAIN's performance against other proposed data-center network designs, such as PortLand [27] or VL2 [17], because these require specific network topologies. SPAIN's support for arbitrary topology is an advantage: one can evaluate or use it on the topology one has access to. (We plan to rebuild our testbed network to support fat-tree topologies, but this is hard to do at scale.) However, in Section 10.5, we compare SPAIN's performance against shortest-path routing, as is done in SEATTLE [22], TRILL [7], and IEEE 802.1aq [5].

10.1 Configuration and workloads

We conducted our evaluation on three racks that are part of the (larger) Open Cirrus testbed [13]. All our experiments are run on 80 servers spread across these three racks (rack 1 has 23 servers, rack 2 has 28, and rack 3 has 29). These servers have quad-core 2GHz Intel Xeon CPUs, 8GB RAM and run Ubuntu 9.04.

Each server is connected to a 3500-series HP 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 SPAIN's benefits, we added 10-GigE cross-connects between the 3500 switches, so that each such switch is connected to another



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

Figure 4: Wiring topology used in our experiments

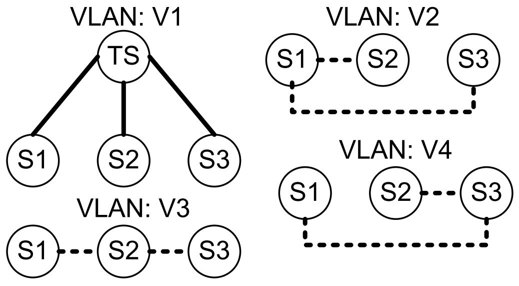


Figure 5: VLANs used by SPAIN for our topology

switch in each physical rack. Fig. 4 shows the resulting wired topology, and Fig. 5 shows the four VLANs computed by the SPAIN offline configuration algorithms.

In our tests, we used a "shuffle" workload (similar to that used in [17]), an all-to-all memory-to-memory bulk transfer among N participating hosts. This communication pattern occurs 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 10 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.

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.

We found that we could not get optimal bi-directional TCP throughput, even for unmodified Linux in a two-host configuration, without using "Jumbo" (9000-byte) Ethernet packets. (We were able to get optimal one-way throughput using 1500-byte packets.) We are not entirely sure of the reason for this problem. The TCP experiments described in this paper all use Jumbo packets.

Table 4: End-host overheads

Configuration	ping RTT (usec)	TCP throughput (Mbit/sec) 1-way	TCP throughput (Mbit/sec) 2-way, [min,max]
Unmodified Linux	98	990	1866 [1858,1872]
1st pkt, cold start	4.03 ms		
SPAIN, no chirping	98	988	1860 [1852,1871]
1st pkt, cold start	3.94 ms		
SPAIN w/chirping	99	988	1866 [1857,1876]
1st pkt, cold start	4.84 ms		

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

Table 4 shows the overhead measurements. The SPAIN driver does not appear to measurably affect TCP throughput or "ping" latency. Note that the chirping protocol does not measurably change either throughput or warm-start latency, even though it adds some data-structure updates on every packet transmission and reception. However, it appears to increase cold-start latency slightly, probably because of the CPU costs of allocating and initializing some data structures.

Table 4 shows throughputs for a single TCP flow in each direction. The shuffle workload, described in Sec. 10.1, sometimes leads to an imbalance in the number of TCP flows entering and leaving a node. We discovered that (even in unmodified Linux) this imbalance can lead to a significant throughput drop for the direction with fewer flows. For example, with 9 flows in one direction and 1 flow in the other, the 1-flow direction only gets 274 Mbps, while the 9-flow direction gets 984 Mbps. We are not sure what causes this.

10.3 SPAIN vs. spanning tree

Table 5 shows how SPAIN compares to spanning tree when running the shuffle workload on the Open Cirrus testbed.

Table 5: Spanning-tree vs. SPAIN

	Spanning Tree	SPAIN
Mean goodput/host (Mb/s)	449.25	834.51
Aggregate goodput (Gb/s)	35.60	66.68
Mean completion time/host	744.57 s	397.50 s
Total shuffle time	831.95 s	431.12 s

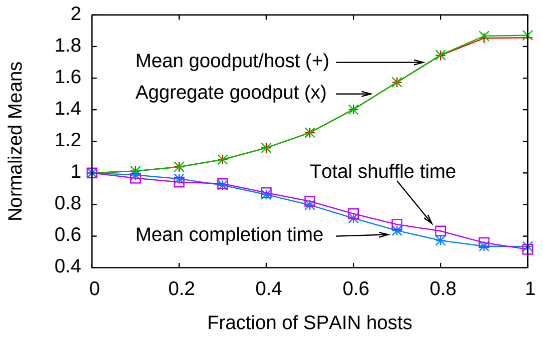
Results are means of 10 trials, 500 MBytes/trial, 80 hosts

Our SPAIN trials yielded an aggregate goodput of 66.68 Gbps, which is 83.35% of the ideal 80-node goodput of 80 Gbps. This is an improvement of 87.30% over the spanning tree topology for the same experiment.

Based on the two-node bidirectional TCP transfer measurement shown in Table 4, the SPAIN goodput should have been (80*1860/2) Mbps or 74.4 Gbps. Thus the observed goodput is about 10% less than this expected goodput. We suspect that the discrepancy is the result of the decreased throughput, described in Sec. 10.2, caused by occasional flow-count imbalances during these experiments. Note that we monitored the utilization of all links during the SPAIN experiments, and did not see any saturated links.

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 10 trials for each fraction f, doing different random node assignments for each trial.



Results are means over 10 trials

Figure 6: Incremental deployability

In Fig. 6, 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 nodes on which SPAIN is deployed. The y-values for each curve are normalized to the 0%-SPAIN results. We saw very little trial-to-trial variation (less than 2.5%) in these experiments, with the exception of total shuffle time, which varies up to 13% between trials. This variation does not seem to depend on the use of SPAIN.

As expected, the aggregate goodput increases and the mean completion times decreases as the fraction of SPAIN nodes increases. The curve for the aggregate goodput increases until and flattens at about $f = 0.9$ at which point none of the links in our network are bottlenecked. Hence, at $f = 0.9$, even flows from or to non-SPAIN nodes do not experience any congestion.

10.5 SPAIN vs. Shortest-Path Routing

Protocols such as SEATTLE [22], TRILL [7], and IEEE 802.1aq [5] improve over the spanning-tree protocol by using Shortest-Path Routing (SPR). Although we did not test SPAIN directly against those three protocols, we can compare SPAIN's performance to SPR-based paths by emulation: we restrict the paths employed

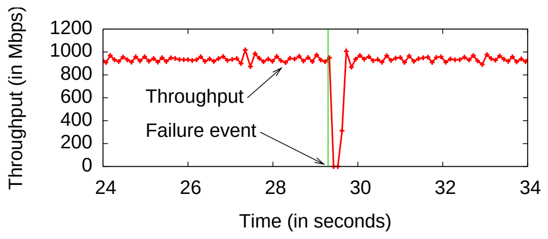


Figure 7: Fault-tolerance experiment

by SPAIN to only those that use the shortest paths between the switches in our test network. In this network, as shown in Fig. 4, the shortest paths between switches S1, S2, and S3 do not go through the core switch (TS); i.e., they do not include the links shown as "VLAN V1" in Fig. 5. (TRILL supports equal-cost multipath, but this would be hard to apply to the topology of Fig. 4.)

We then re-ran the shuffle experiment using the SPR topology, and achieved an aggregate goodput of 62.28 Gbps, vs. SPAIN's 66.61 Gbps goodput. The total shuffle time for SPR is 512.73 s, vs. SPAIN's 430 s. As mentioned in Sec. 10.3, SPAIN's throughput is limited by CPU overheads, so the relatively minor improvement of SPAIN over SPR may be a result of these overheads.

We note that, regardless of the relative performance of SPAIN and SPR, SPAIN retains the advantage of being deployable without any changes to switches. TRILL, SEATTLE, and IEEE 802.1aq Shortest Path Bridging will all require switch upgrades.

10.6 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. 7 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 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 87% on a testbed that would not support most other scalable-Ethernet designs.

We recognize that significant additional work could be required to put SPAIN into practice in a large-scale network. This work includes the design and implementation of a real-time central controller, to support dynamic global re-balancing of link utilizations, and also improvements to SPAIN's end-host mechanisms for assigning flows to VLANs. We also do not fully understand how SPAIN will affect broadcast loads in very large networks.

Acknowledgements

We would like to thank our HP colleagues Peter Haddad, Jean Tourrilhes, Yoshio Turner, Sujata Banerjee, Paul Congdon, Dwight Barron, and Bob Tarjan; the reviewers, and our shepherd, Jennifer Rexford.

References

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

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

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

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

[5] IEEE 802.1aq - Shortest Path Bridging. `http://www.ieee802.org/1/pages/802.1aq.html`.

[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. Aron and P. Druschel. Soft timers: efficient microsecond software timer support for network processing. In *Proc. SOSP*, 1999.

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

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

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

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] John D'Ambrosia. IEEE P802.3ba Task Force Timeline. General Information Session at Interim Meeting `http://www.ieee802.org/3/ba/public/jan10/agenda_01_0110.pdf`, January 2010.

[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] J. Mudigonda, P. Yalagandula, M. Al-Fares, and J. C. Mogul. SPAIN: Design and Algorithms for Constructing Large Data-Center Ethernets from Commodity Switches. Tech. Rep. HPL-2009-241, HP Labs, 2009.

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

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

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

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

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

[33] 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.