PAST: Scalable Ethernet for Data Centers

Brent Stephens Alan Cox
Rice University
Houston, TX
brents,alc@rice.edu

Wes Felter Colin Dixon John Carter
IBM Research
Austin, TX
wmf,ckd,retrac@us.ibm.com

ABSTRACT

We present PAST, a novel network architecture for data center Ethernet networks that implements a *Per-Address Spanning Tree* routing algorithm. PAST preserves Ethernet's self-configuration and mobility support while increasing its scalability and usable bandwidth. PAST is explicitly designed to accommodate unmodified commodity hosts and Ethernet switch chips. Surprisingly, we find that PAST can achieve performance comparable to or greater than Equal-Cost Multipath (ECMP) forwarding, which is currently limited to layer-3 IP networks, without *any* multipath hardware support. In other words, the hardware and firmware changes proposed by emerging standards like TRILL are not required for high-performance, scalable Ethernet networks. We evaluate PAST on Fat Tree, HyperX, and Jellyfish topologies, and show that it is able to capitalize on the advantages each offers. We also describe an OpenFlow-based implementation of PAST in detail.

Categories and Subject Descriptors

C.2 [**Internetworking**]: Network Architecture and Design

General Terms

Algorithms, Design, Management, Performance

Keywords

Software Defined Networking, OpenFlow, Data Center

1. INTRODUCTION

The network requirements of modern data centers differ significantly from traditional networks, so traditional network designs often struggle to meet them. For example, layer-2 Ethernet networks provide the flexibility and ease of configuration that network operators want, but they scale poorly and make poor use of available bandwidth. Layer-3 IP networks can provide better scalability and bandwidth, but are less flexible and are more difficult to configure and manage. Network operators want the benefits of both designs, while at the same time preferring commodity hardware over expensive custom solutions to reduce costs. Thus, our challenge is to provide the ease of use and flexibility of Ethernet and the scalability and performance of IP using only inexpensive commodity hardware.

More precisely, a modern data center network should meet the following four functional requirements [12, 23, 33, 37].

1. **Host mobility:** Hosts—especially virtual hosts—must be movable without interrupting existing connections or requiring address changes. *Live migration is needed to tolerate faults and achieve high host utilization.*
2. **Effective use of available bandwidth:** A workload should not be limited by network bandwidth while usable bandwidth exists along alternate paths. *Path selection that prevents using available bandwidth reduces cost-effectiveness.*
3. **Self-configuration:** Network elements, e.g., routers or (v-)switches, must be able to forward traffic without manual configuration of forwarding tables. *At scale, manual configuration makes management untenable.*
4. **Scalability:** The network should scale to accommodate the needs of modern data centers without violating the preceding requirements. *Scaling by hierarchically grouping smaller networks, e.g., grouping Ethernet LANs via IP, may not satisfy our requirements.*

In addition to these functional requirements, we limit our design space to architectures that can be implemented and managed efficiently with commodity hardware and software, which leads to three additional design requirements:

1. **No hardware changes:** The architecture must work with commodity networking hardware. *Architectures that require proprietary hardware are harder to deploy and lose the advantages offered by economies of scale.*
2. **Respects layering:** The architecture must work with unmodified software stacks, e.g., operating systems and hypervisors, and higher layers must not need to understand details of the architecture's implementation. *Customers have made large investments in their current software stacks and will resist adopting a network architecture that breaks them.*
3. **Topology independent:** The architecture must work with arbitrary topologies, e.g., Fat Tree, HyperX, or Jellyfish. *Restricting the network to a particular topology, e.g., Fat Tree, prevents network operators from considering alternate topologies that may offer better performance, lower cost, or both.*

Table 1 compares existing data center network architectures and recent academic work against the above requirements. We can see that no existing architecture meets all of them. One reason is that the requirements often conflict with one another.

CoNEXT'12, December 10–13, 2012, Nice, France.

Architecture	Functional Requirements				Design Requirements		
	Mobility	High BW	Self Config	Scales	No H/W Changes	Respect Layers	Topo Ind
Ethernet with STP	✓	X	✓	X	✓	✓	✓
IP (e.g. OSPF)	X	✓	X	✓	✓	✓	✓
MLAG [29]	✓	✓	✓	X	✓	✓	✓
SPAIN [30]	✓	✓	✓	X	✓	X	✓
PortLand [33]	✓	✓	✓	✓	✓	✓	X
VL2 [14]	X	✓	X	✓	✓	X	X
SEATTLE [23]	✓	X	✓	✓	✓	✓	✓
TRILL [36]	✓	✓	✓	X	X	✓	✓
EthAir [37], VIRO [21]	✓	X	✓	✓	✓	✓	✓
PAST	✓	✓	✓	✓	✓	✓	✓

Table 1: Comparison of data center network architectures.

For example, Ethernet's distributed control protocol provides host mobility with little or no configuration. However, it does not scale well beyond roughly one-thousand hosts due to its use of broadcast for name resolution. Further, it makes poor use of available bandwidth because it uses a single spanning tree for packet forwarding—a limitation imposed to avoid forwarding loops.

To address these problems, current large data center networks connect multiple Ethernet LANs using IP routers [13] and run scalable routing algorithms over a smaller number of IP routers. These layer-3 routing algorithms allow for shortest path and Equal-Cost Multipath (ECMP) routing, which provide much more usable bandwidth than Ethernet's spanning tree. However, the mixed layer-2/layer-3 solution requires significant manual configuration and (typically) limits host mobility to be within a single LAN.

The trend in recent work to address these problems is to introduce special hardware and topologies. For example, PortLand [33] is only implementable on Fat Tree topologies and requires ECMP hardware, which is not available on every Ethernet switch. TRILL [36] introduces a new packet header format and thus requires new hardware and/or firmware features.

We pose the following question: Are special hardware or topologies necessary to implement a data center network that meets our requirements, or can we build such a data center network with only commodity Ethernet hardware?

Surprisingly, we find that we *can* build a data center network that meets all of the requirements using only the most basic Ethernet switch functionality. Contrary to the suggestions of recent work, special hardware and restricted topologies are *not* necessary.

To prove this point, we present PAST, a flat layer-2 data center network architecture that supports full host mobility, high end-to-end bandwidth, autonomous route construction, and tens of thousands of hosts on top of common commodity Ethernet switches. PAST satisfies our functional and design requirements as follows. When a host joins the network or migrates, a new spanning tree is installed to carry traffic destined for that host. This spanning tree is implemented using only entries in the large Ethernet (exact match) forwarding table present in commodity switch chips, which allows PAST to support as many hosts as there are entries in that table. In aggregate, the trees spread traffic across all links in the network, so PAST provides aggregate bandwidth equal to or greater than layer-3 ECMP routing. PAST provides Ethernet semantics and runs on unmodified switches and hosts without appropriating the VLAN ID or other header fields. Finally, PAST works on arbitrary network topologies, including HyperX [2] and Jellyfish [38], which can perform as well as or better than Fat Tree [33] topologies at a fraction of the cost.

PAST can be implemented in either a centralized or distributed fashion. We prefer a centralized software-defined network (SDN) architecture that computes the trees on a high-end server processor rather than using the underpowered control plane processors present in commodity Ethernet switches. Our OpenFlow-based PAST implementation was crafted carefully to utilize the kinds of match-action rules present in commodity switch hardware, the number of rules per table, and the speed with which rules can be installed. By restricting PAST to route solely using destination MAC addresses and VLAN tags, we can use the large layer-2 forwarding table, rather than relying on the more general, but much smaller, TCAM forwarding table, as is done in previous OpenFlow architectures.

The main contributions of this paper are as follows:

1. We present PAST, a novel network architecture that meets all of the requirements described above using a per-address spanning tree routing algorithm.
2. We present an implementation that makes efficient use of the capabilities of commodity switch hardware.
3. We evaluate PAST on Fat Tree, HyperX and Jellyfish topologies and show that it can make full use of the advantages that each offers. We also offer the first comparison of the HyperX and Jellyfish topologies.

The remainder of the paper is organized as follows. In Section 2 we present background information on switch hardware and routing. We describe the design of the PAST routing algorithm in Section 3 and its implementation in Section 4. In Section 5 we present the experimental methodology that we use to evaluate PAST, describe the topologies and workloads that we use in our evaluation, and present the results of our evaluation. We describe the previous work that most influenced PAST in Section 6. Finally, in Section 7 we draw conclusions and present ideas for future work.

2. BACKGROUND

This section describes how current commodity Ethernet forwarding hardware works and discusses the state-of-the-art in data center network routing.

2.1 Switch Hardware Overview

While many vendors produce Ethernet forwarding hardware, the hardware tends to exhibit many similarities due in part to the trend of using "commodity" switch chips from vendors such as Broadcom and Intel at the core of each switch. In the following discussion, we focus on one such switch chip, the Broadcom StrataXGS

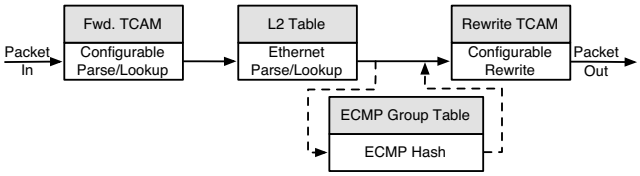

Figure 1: Partial Broadcom Switch Pipeline

Table	Broadcom Trident	HP ProVision	Intel FM6000	Mellanox SwitchX
TCAM	∼2K + 2K	1,500	24K	0?
L2/Eth	∼100K	∼64K	64K	48K
ECMP	∼1K	unknown	0	unknown

Table 2: 10 Gbps Ethernet Switch Table Sizes (number of entries)

BCM56846 [6] ("Trident"), and the IBM RackSwitch G8264 top-of-rack switch [17] that uses the Trident chip at its core. We believe this design is representative of 10 Gigabit Ethernet switches with the best price/performance currently available on the market. While our discussion focuses on a particular switch and switch chip, our work exploits chip features and design tradeoffs that are common in modern switches. Our choice of Trident and G8264 was driven by what we have available and the fact that the G8264 firmware exposes the ability for OpenFlow to install rules in the L2 table.

The rise of commodity switch chips is well-known in the networking community [25], but chip vendors typically provide only short data sheets with few details to the public; the specific details of the switch firmware are proprietary. This lack of information about switch internals makes it difficult for networking researchers to consider the constraints of real hardware. If a new switch capability is needed to enable a research idea, it is difficult to estimate whether it is available in current firmware, can be added with just firmware changes, or requires hardware changes. Similarly, designing forwarding mechanisms without understanding the size, cost, and functionality of the switch forwarding tables can lead to inefficient or non-scalable designs.

2.1.1 The Trident Switch Chip

Figure 1 presents a high-level overview of the relevant portion of the Trident packet processing pipeline. Each box represents a table that maps packets with certain header fields to an action. Each table differs in which header fields can be matched, how many entries it holds, and what kinds of actions it allows, but all tables are capable of forwarding packets at line-rate. Typical actions include sending a packet out a specific port or forwarding it to an entry in another table. The order in which tables can be traversed is constrained; the allowed interactions are shown with directed arrows. Table 2 presents the approximate size of each of these tables for several commodity Ethernet switch chips. The Trident pipeline contains dozens of other configurable tables to support features such as IP routing, DCB, MPLS, and multicast, but we do not discuss those tables further.

The L2 (or Ethernet) table performs an exact match lookup on two fields: VLAN ID and destination MAC address. It is by far the largest table, having tens of thousands of entries, because it can be implemented with SRAM. The output of the L2 table is either an output port or a *group*, which can be thought of as a virtual port used to support multipathing or multicast. If the action for a packet is to output it to an ECMP group, the switch hashes configurable header fields (usually source and destination IP address and port number) to select a port through which the packet should be forwarded. ECMP allows traffic to be load balanced across multiple paths between hosts. Traditionally, ECMP uses minimum hop count paths, but the hardware does not enforce this. Trident can be configured to support one thousand ECMP groups, each with four ports.

The rewrite and forwarding TCAMs (Ternary Content Addressable Memories) are tables that can wildcard match on most packet header fields, including per-bit wildcards. The rewrite TCAM supports output actions that modify packet headers, while the forwarding TCAM is used to more flexibly choose an output port or group. The greater flexibility of TCAMs comes at a cost. They consume much higher area and power per entry than SRAM. Therefore on-chip TCAM sizes are typically limited to a few thousand entries.

2.1.2 Switch Control Plane

The switch chip is not a general purpose processor, so switches typically also contain a control plane processor that is responsible for programming the switch chip, providing the switch management interface, and participating in control plane protocols such as spanning tree or OSPF. In a software-defined network, the control processor also translates controller commands into switch chip state.

Unique among current switches, the G8264's OpenFlow 1.0 implementation allows OpenFlow rules to be installed in the L2 table. Specifically, if it receives a rule that exact matches on (only) the Destination MAC address and VLAN ID, it installs the rule in the L2 table. Otherwise it installs the rule in the appropriate TCAM, as is typical of OpenFlow implementations.

In traditional Ethernet, much of the forwarding state is learned automatically by the switch chip based on observed packets. A software defined approach shifts some of this burden to the control plane processor and controller, adding latency and potential bottlenecks.

To provide an OpenFlow control plane performance baseline, we characterize the G8264's performance with custom microbenchmarks using the OFlops [34] framework. The G8264 can install 700-1600 new rules per second, and each rule installation takes 2-12ms, depending on how many rules are batched in each request. Also, each time the switch receives a packet for which no forwarding rule exists, the switch must generate a message to the controller. If the switch receives more than 200 packets per second for which no route exists, the control plane becomes saturated, which can result in message latency on the order of seconds and message losses. This is problematic, since data centers often operate under tight SLAs on the order of 10-100ms [5], and violating these SLAs can have serious consequences, such as decreasing the quality of results in the case of Google [5] and reducing sales in the case of Amazon [15].

These measurements convince us that reactive forwarding rule installation will not provide acceptable performance, at least with the G8264's current control plane processor. Thus, we adopt the approach of DevoFlow [7] whereby routes are eagerly computed and installed before their first use. It is worth nothing that eager routing does not prevent hosts, switches, or links from dynamically being added and removed from the network. Rather, eager routing means that corresponding routes are updated eagerly when any such network events occur.

Algorithm 1 – Per-Address Spanning Tree (PAST)

Input: network topology $G = (V, E)$ and sets $H, S \subseteq V$, where H is the set of hosts and S is the set of switches
Output: a forwarding table T_s for each switch $s \in S$
begin
Initialize: $\forall s \in S, T_s = \{\}$
Define: $BFS\_ST(h, G)$ returns a shortest path spanning tree of G, rooted at h
Define: $v.parent\_edge(Tree)$ returns the edge in the tree $Tree$ that connects v to its parent
for all $h \in H$ **do**
  $G_{st} = BFS\_ST(h, G)$
  for all $s \in S$ **do**
    $T_s[h] = s.parent\_edge(G_{st})$
  end for
end for

2.2 Routing Design Space

Generally speaking, there are two approaches to scalable routing. The first entails making addresses topologically significant so routes can be aggregated in routing tables. The second is simply to have enough space in routing tables to allow for all routable addresses to have at least one entry.

As described above, the two forwarding tables (Ethernet and TCAM) differ in size by roughly two orders of magnitude. Given its small size, any routing mechanism that requires the flexibility of a TCAM for matching must aggregate routes, otherwise the few thousand TCAM entries per switch will be quickly exhausted. The larger size of the Ethernet forwarding table means that any forwarding mechanism that matches only on Destination MAC and VLAN ID can fit one entry per routable address per switch, even for large networks. Note that aggregation cannot be used with the Ethernet forwarding table as it allows for exact matching only.

Previous SDN proposals employ TCAM rules, so they have been forced to use aggregate routes [3, 33]. Aggregating routable addresses means that either the topology must be constrained or a virtual topology must be created on top of the physical one, which introduces inefficiencies. For example, PortLand [33] constrains the topology to a Fat Tree and assigns addresses based on each node's position in the tree, allowing for aggregation at each level. Virtual ID Routing [26] and Ethernet on Air [37] both build a tree and hierarchically assign addresses within the tree. While this approach works on arbitrary topologies, it does so by disabling some links and introducing paths that are up to a factor of two longer than necessary.

In contrast, the traditional Ethernet spanning tree protocol (STP), its would-be successor TRILL [36], and PAST place rules in the Ethernet forwarding table and exploit its larger size to have one entry per routable address in each switch. As a result, these routing algorithms work on arbitrary topologies, including ones that offer better price-performance than Fat Tree, like Jellyfish and HyperX. However, STP does not exploit the potential advantages of such topologies because, to avoid routing loops, it forwards all traffic along a single tree. Extensions to STP that allow one spanning tree per VLAN (as in SPAIN [30]) and vendor-specific techniques like multi-chassis link aggregation (MLAG) [29], which creates a logical tree on top of a physical mesh, can mitigate some, but not all, of STP's routing inefficiency. In an enterprise or cloud environment, VLAN IDs are needed to support security and traffic isolation, and thus should not be used for normal routing. TRILL generalizes these approaches by running IS-IS to build shortest path routes between switches. All of these approaches, except PAST, use broadcast for address resolution, limiting their scalability.

Two orthogonal extensions to routing are commonly used to fully exploit available bandwidth: *multipath routing* and *Valiant load balancing*. ECMP allows traffic between two hosts to use any minimal path, increasing path diversity and decreasing the likelihood of artificial 'hot spots' in the network where two flows collide even though non-colliding paths exist. Because ECMP requires there to be multiple paths, it has only been possible on architectures that can find all shortest paths, such as IP routing and TRILL. Valiant load balancing increases path diversity by using non-minimal paths. In Valiant routing, traffic is first forwarded minimally to a random switch and then follows the minimal path to its destination. This design also helps avoid artificial hot spots.

3. PAST DESIGN

As Table 1 shows, no existing architecture meets the requirements laid out in Section 1. PAST fills this gap by providing traditional Ethernet benefits of self-configuration and host mobility while using all available bandwidth in arbitrary topologies, scaling to tens of thousands of hosts, and running on current commodity hardware. PAST does so by installing routes in the Ethernet table without the constraints of STP. PAST is a previously unexplored point in the design space.

3.1 PAST Routing

PAST's design is guided by the structure of commodity switches' Ethernet forwarding tables. Any routing algorithm that can express its forwarding rules as a mapping from a <Destination MAC addr, VLAN ID> pair to an output port can be implemented using the large Ethernet forwarding table. By design, it is possible to represent an arbitrary spanning tree using rules of this form. The Ethernet table was designed to support the traditional spanning tree protocol (STP), but we observe that it can also implement a separate spanning tree per destination host, which results in PAST. It is possible to construct a spanning tree for any connected topology, so PAST is topology-independent.

The topologies we consider have high path diversity, so many possible spanning trees can be built for an address. Each individual tree uses only a fraction of the links in the network, so it is beneficial to make the different trees as disjoint as possible to improve aggregate network utilization. The literature is rife with spanning tree algorithms—in this paper we explore several alternatives and we plan to explore more in the future.

3.1.1 Baseline PAST

For our baseline PAST design, we build destination-rooted shortest-path spanning trees. The intuition behind this design is that shortest-path trees reduce latency and minimize load on the network. We employ a breadth-first search (BFS) algorithm to construct the shortest-path spanning trees, as shown in Algorithm 1. A BFS spanning tree is built for every address in the network. This spanning tree, rooted at the destination, provides a minimum-hop-count path from any point in the network to that destination. Any given switch only uses a single path for forwarding traffic to the host, and the paths are guaranteed to be loop-free because they form a tree. No links are ever disabled. Because a different spanning tree is used for each destination, the forward and reverse paths between two hosts in a PAST network are not necessarily symmetric.

When building each spanning tree, we often have multiple options for the next-hop link. The way that the next-hop link is selected may impact path diversity, load balance, and performance. We evaluated two options, one that simply picks a uniformly random next-hop and one that employs weighted randomization. We refer to these two baseline designs as PAST-R (random) and

PAST-W (weighted), respectively. PAST-R performs breadth-first search with random tie-breaking. Intuitively, this causes the spanning trees to be uniformly distributed across the available links. However, not all links in a spanning tree are the same—links closer to the root are likely to carry more traffic than links lower in the tree. Thus PAST-W weights its random selection by considering how many hosts (leaves) each next-hop switch has as children, summed across all spanning trees built so far.

PAST does not care whether an address (MAC address-VLAN pair) represents a VM, a physical host, or a switch. This is the choice of the network operator. Since PAST can support tens of thousands of addresses on commodity hardware, there is no need to share, rewrite, or virtualize addresses in a network. Likewise, a host may use any number of addresses if it wishes to increase path diversity at the cost of forwarding state.

PAST has similarities to ECMP. ECMP enables load-balancing across minimum-hop paths at per-flow granularity. PAST enables load-balancing across minimum-hop paths at per-destination granularity. As the number of destinations per switch increases relative to the number of minimum-cost paths, we expect the performance of PAST to approach that of ECMP. We show this is the case in practice in Section 5.

3.1.2 *Non-minimal PAST*

As noted in Section 2.2, some topologies, e.g., HyperX, require non-minimal routing algorithms like Valiant routing to achieve high performance under adversarial workloads. To support these topologies, we implemented a variant of the baseline PAST algorithm that selects a random intermediate switch i as the root for the BFS spanning tree for each host h. The switches along the path in the tree from i to h are then updated to route towards h, not i, so that h is the sink of the tree. We refer to this approach as NM-PAST (non-minimal PAST). As with the baseline algorithm, we implemented both random (NM-PAST-R) and weighted random (NM-PAST-W) variants.

NM-PAST is inspired by Valiant load balancing. In Valiant load balancing, all traffic is first sent through randomly chosen intermediate switches. Similarly, most traffic in NM-PAST sent to h will first be sent through the randomly chosen switch i. Only the hosts along and below the path in the tree from i to h do not forward traffic through i.

3.2 Discussion

Broadcast/Multicast: PAST is currently intended only for unicast traffic. We treat unicast and multicast routing as orthogonal features; it is possible to simultaneously use PAST for unicast and some other system such as STP for multicast. Both traditional solutions, such as STP, and novel solutions, such as building multicast and broadcast groups with SDN, are compatible with PAST. We believe it is possible to optimize multicast traffic by building a separate multicast distribution tree for each multicast address, but we do not explore this possibility. Additionally, if performance isolation of unicast from broadcast and multicast traffic is desired, the network can ensure that the unicast spanning trees do not use any of the links used for broadcast and multicast.

Security: PAST does not reuse or rewrite the Ethernet VLAN header, so VLANs can be employed for security and traffic isolation, as in traditional Ethernet.

Flow Splitting: In order to benefit from flow splitting, such as MPTCP [39], the network is required to offer multiple paths to a destination. This is possible with PAST because the Ethernet forwarding table matches on the <Destination MAC addr, VLAN ID> pair. If hosts are configured as members of multiple VLANs, then MPTCP can perform flow splitting across VLANs. In fact, the probability of benefiting from flow splitting is greater in PAST than in ECMP because it is possible to actively try to build edge disjoint spanning trees for each VLAN of an address, whereas hash collisions are possible in ECMP.

Virtualization: As stated earlier, PAST provides standard Ethernet semantics with no need for hosts to understand any of PAST's implementation details. As a consequence, any higher layer, including network virtualization overlays like NetLord [32], SecondNet [16], MOOSE [28], and VXLAN [27], can operate seamlessly atop PAST.

Live VM migration is an expected feature in virtualized clusters, and PAST must be able to update the tree for the migrating host quickly to avoid delaying the migration. Both Xen and VMware send a gratuitous ARP from the VM's new location during migration, effectively notifying the controller that the it should reroute traffic for that VM. As we discuss in Section 4, updating a single tree in PAST is expected to take less than 20ms, which is comparable to the existing pause time involved in VM migration.

4. PAST IMPLEMENTATION (SDN)

A network architecture requires more than a routing algorithm to meet the requirements laid out earlier. In this section we describe other aspects of PAST, including address detection, address resolution, broadcast/multicast, topology discovery, route computation, route installation, and failure recovery.

We implemented the PAST architecture as an extension to the Floodlight [11] OpenFlow controller and a collection of IBM RackSwitch G8264 switches. While PAST should work with any OpenFlow 1.0 compliant switch, to the best of our knowledge the G8264 is the only switch that currently supports installing OpenFlow rules in the Ethernet forwarding table. Our implementation falls back to putting entries into the (much smaller) TCAM table if there is no way to access the larger Ethernet forwarding table, but this limits the scalability of the implementation. By using only the Ethernet forwarding table, the TCAM table(s) can be used for other purposes such as ACLs and traffic engineering.

Address detection: Our controller configures each switch to snoop all ARP traffic and forward it to the controller. The gratuitous ARPs that are generated on host boot and migration provide timely notification of new or changed locations and trigger (re)computation of the spanning tree for the given address. The controller also tracks the IP addresses associated with each address so that it can respond to ARP requests.

Address resolution: Our implementation eliminates the scaling problems of flooding for address resolution by using the controller for address resolution, specifically ARP. Broadcast ARP packets are encapsulated in `packet_in` messages and sent to the controller, which responds to the request. Additional protocols that require broadcast to operate, such as DHCP, IPv6 Neighbor Discovery, and Router Solicitation, can also be intercepted and handled by the network controller, although we did not need these protocols in our current implementation. This kind of interposition is technically a layering violation since Ethernet is normally oblivious to higher-layer protocols, but it increases scalability for large networks. However, we note that this interception is an optimization that is not required for correctness and a network operator could forgo it at the cost of scalability.

Broadcast/Multicast Our current prototype focuses on unicast and thus we have not implemented optimized versions of broadcast and multicast. We currently fall back to the Floodlight implementation of multicast and broadcast which treats both as broadcast and forwards them using a single spanning tree for the entire network.

In the future we intend to support multiple spanning trees to handle broadcast and multicast traffic extending PAST's benefits to them as well.

Topology discovery: We use Floodlight's built-in topology discovery mechanism, which sends and receives Link Layer Discovery Protocol (LLDP) messages on each port in the network using OpenFlow `packet_out` and `packet_in` messages. LLDP messages discover whether a link connects to another switch and, if it is a switch, the other switch's ID.

Route computation: Upon discovering a new (or migrated) address, PAST (re)computes the relevant tree. The time for the controller to compute the tree does not bottleneck the system. In our implementation, computing a single tree with a cold cache on topologies with 8,000 and 100,000 hosts takes less than 1ms and 5ms, respectively. If multiple trees are computed and the cache is warm, the time decreases to less than 40μs and 500μs, respectively, and multiple cores can be used to increase throughput because computation of each tree is independent and thus trivially parallel. For example, on a single core we are able to compute trees for all hosts in the network in approximately 300ms for an 8,000 host network and 40 seconds for a 100,000 host network. Loosely speaking, our computation scales linearly with the number of switch-to-switch links in the network and in the limit can create a tree in one μs per 300 links in the network on a single core of a 2.2 GHz Intel Core i7 processor.

Route installation: Whenever a tree is (re)computed, PAST pipelines the installation of the relevant rules on the switches. Once a switch has received an installation request, installing the rule takes less than 12ms. To ensure a rule is placed in the Ethernet forwarding table, our switches require that the rules specify an exact match on destination MAC address and VLAN (only) and have priority 1000. While in theory it is possible that installing a recomputed tree could create a temporary routing loop, we have yet to observe this in practice. This problem could be prevented at a cost in latency by first removing rules associated with trees being replaced and issuing an OpenFlow barrier to ensure they are purged, before installing new trees.

Failure Recovery Failures are common events in large networks [14] and should be handled efficiently. While it has been sufficient for our current implementation to naively recompute all affected trees when switches and links leave and join the network, more scalable incremental approaches are possible. For example, when a new switch is added to the network, it can be initially added as a leaf node to all existing trees so that only the new switch needs to be updated with the existing hosts. Similarly, only portions of trees will be affected by switch and link failures, and it is possible to patch the trees to restore connectively to the affected hosts without disturbing the unaffected traffic. It is worth noting that new links appearing do not affect any existing trees, but they are of no benefit until they are incorporated into a tree. We rebuild random trees at regular intervals to gradually exploit new links and re-optimize our trees.

The bottleneck in failure recovery is installing flow entries; updating 100K trees would take over a minute. Optimizing flow installation is thus a critical concern in OpenFlow switches.

Other details: Our current PAST implementation consists of approximately 4,000 lines of Java. Most code belongs to a few modules that implement tree computation and address resolution, but we also made modifications to the Floodlight core to install trees instead of simple paths and to place rules in the Ethernet table rather than the TCAM. Our controller connects to all switches using a separate (out-of-band) control network that is isolated from the data network. This isolation allows PAST to bootstrap the network quickly and recover from failures that could partition the data network.

5. EVALUATION

In this section, we describe a set of simulation experiments that we use to compare PAST's performance and scalability against several existing network architectures. We find that it performs equal to or better than existing architectures, including ECMP on Fat Tree (e.g., PortLand) and ECMP on arbitrary topologies (e.g., TRILL). We have a working implementation of PAST, but our testbed only includes 20 servers with a total of 80 NICs, necessitating simulation to evaluate at scale.

We first describe our experimental methodology, including brief descriptions of our simulator, workload data, and topologies. We then provide a short comparison of the three topologies that we evaluated (Fat Tree, HyperX, and Jellyfish), which confirms prior findings that HyperX and Jellyfish topologies can offer better performance than Fat Trees at lower cost. This work also represents the first comparison of the HyperX and Jellyfish topologies. Finally, we evaluate the topology-independent PAST algorithm and its variants. While we do not find a significant difference between the uniform and weighted random PAST variants, our results show that PAST performs as well as ECMP on all topologies under uniform random workloads and NM-PAST performs better than ECMP routing and Valiant load-balancing under adversarial workloads.

5.1 Methodology

We built a working implementation of PAST running on four IBM RackSwitch G8264 10GbE switches to validate the feasibility of our design. Since our testbed only includes 20 servers with a total of 80 NICs, we use simulation to generate the results presented below.

5.1.1 Simulator

To evaluate issues that arise at scale and to explore the scaling limits of various designs, we wrote a custom discrete event network simulator. Our network simulator can replay flow traces in open-loop mode or programmatically generate flows in closed-loop mode. For performance, the simulator models flows instead of individual packets, omitting the details of TCP dynamics and switch buffering. The bandwidth consumption of each flow is simulated by assuming that each TCP flow immediately receives its max-min fair share bandwidth of the most congested link that it traverses. We simulate three data center network topologies on four different workloads to compare the performance of different forwarding algorithms that can be implemented with current Ethernet switch chips.

The simulator uses Algorithm 2 to compute the rate of the TCP flows in the network. This algorithm was first used by Curtis *et al*. in the DevoFlow [7] network simulator. The end result of this algorithm is that each flow receives its fair share of bandwidth of the most congested port it traverses.

5.1.2 Workloads

We use four different workloads in our evaluation.

The first two workloads represent adversarial (*Stride*) and benign (*URand*) communication patterns. On a network with N hosts, the Stride-s workload involves each host with index i sending data to the host with index $(i + s)mod(N)$. In the URand-u workload, each host sends data to u other hosts that are selected with uniform probability.

Algorithm 2 – Flow rate computation – Adapted from DevoFlow [7]

Input: a set of flows F, a set of ports P, and a rate $p.rate()$ for each $p \in P$. Each $p \in P$ is a set of flows such that $f \in p$ iff flow f traverses through port p.
Output: a rate $r(f)$ of each flow $f \in F$

```
begin
Initialize: F_a = ∅; ∀f, r(f) = 0
Define: p.used() = Σ_{f∈F_a∩p} r(f)
Define: p.unassigned_flows() = p − (p ∩ F_a)
Define: p.flow_rate() =
  (p.rate() − p.used())/|p.unassigned_flows()|
while P ≠ ∅ do
  p = arg min_{p∈P} p.flow_rate()
  P = P − p
  rate = p.flow_rate()
  for all f ∈ p.unassigned_flows() do
    r(f) = rate
    F_a = F_a ∪ f
  end for
end while
```

The next two workloads are representative of data center communication patterns. The first data center workload is a closed-loop data shuffle based on MapReduce/Hadoop communication patterns. Inspired by Hedera [4] and DevoFlow [7], we examine a synthetic workload designed to model the shuffle phase of a map-reduce analytics workload. In this *Shuffle* workload, each host transfers 128 MB to every other host, maintaining k simultaneous connections. In this workload, every host is constantly adding load to the network. For our simulations, we set $k = 10$.

The second data center workload is generated synthetically from statistics published about traffic in a data center at Microsoft Research (MSR) by Kandula *et al.* [22], who instrumented a 1500-server production cluster at MSR for two months and characterized its network traffic. We generated synthetic traces based on these characteristics to create the *MSR* workload. Specifically, we sample from the number of correspondents, flow size, and flow inter-arrival time distributions for intra-rack and entire cluster traffic. Because of the inter-arrival time distribution, it is possible for hosts to be idle during parts of the simulation.

5.1.3 Topologies

We evaluate three different data center topologies: *EGFT* (extended generalized fat trees[35]), *HyperX*, and *Jellyfish*. We also evaluate the *Optimal* topology, an unrealistic topology consisting of a single, large, non-blocking switch. We build all three data center topologies using 64-port switches.

While much of the research literature focuses on full-bisection-bandwidth networks, most large-scale data center networks employ some degree of *oversubscription*. Thus, we consider a range of oversubscribed networks for each topology, ranging from $1:1$, which represents a full-bisection network, to $1:5$, which represents a network with bisection bandwidth one-fifth that of a full-bisection network.

When comparing topologies, we simulate instances with equal bisection bandwidth ratio (often referred to as the *oversubscription ratio*). Informally, the bisection bandwidth ratio of a graph is the ratio of the bandwidth of the links that cross a cut of the network to the bandwidth of the hosts on one side of the cut, for the worst case cut of the network that splits the network in half. Formally, the bisection bandwidth ratio of a network $G = (V, E)$, adapted from the definition given by Dally and Towles [8], is:

$$bisec(G) = \min_{S \subseteq V} \frac{\sum_{e \in \delta(S)} w(e)}{\min\{\sum_{v \in S} r(v), \sum_{v \in \bar{S}} r(v)\}}$$

where $\delta(S)$ is the set of edges with one endpoint in S and another in $\bar{S}$, $r(v)$ is the total bandwidth that vertex v can initiate or receive, $w(e)$ is the bandwidth of edge e, and $|\bar{S}| \leq |S| \leq |\bar{S}| + 1$. The bisection bandwidth ratio is a useful metric for comparing topologies because it bounds performance if routing is optimal [8]. Different topologies typically require different numbers of switches and inter-switch links to achieve a particular bisection bandwidth, but the bisection bandwidth for all of the topologies we evaluate can be calculated directly from the properties of the topology. We refer the reader to the references for the specific equations for each topology.

EGFT: Simple fat tree and extended generalized fat tree (EGFT[35]) topologies have long been used in supercomputer interconnects. In recent years they have been proposed for use in Ethernet-based commercial data center networks [3, 9, 33]. One compelling property of EGFT topologies is that they have similar performance on all traffic patterns.

HyperX: The HyperX topology [2] and its less general form, the Flattened Butterfly topology, have also made the transition from supercomputing to Ethernet in recent papers [1]. HyperX topologies are known to make better use of available bisection bandwidth than Fat Trees, and thus can support certain traffic patterns at a lower cost than EGFT. For example, a $1:2$ oversubscribed HyperX can forward uniformly distributed traffic from every host at full line-rate, whereas a $1:2$ oversubscribed EGFT can only forward traffic from every host at half line-rate [24].

Jellyfish: The recently-proposed Jellyfish topology [38] connects switches using a regular random graph. The properties of such graphs have been extensively studied and are well understood. Like HyperX, it can efficiently exploit available bandwidth.

A unique strength of the Jellyfish topology is its flexibility. A Jellyfish network can be upgraded by simply adding switches and performing localized rewiring. On a Jellyfish topology, adding a switch with k inter-switch links only requires moving k other links in the network. In contrast, upgrades to EGFT and HyperX networks tend to be disruptive and/or only efficient for certain network sizes. Additionally, switches in a Jellyfish network do not need to have the same radix, which allows multiple switch generations to coexist in a single data center. Both upgrades and equipment failures preserve the Jellyfish's lack of structure, and the Jellyfish is very cost-effective since all switch ports are utilized, which maximizes use of expensive hardware.

Optimal: The optimal topology is a fictitious network modeled by attaching all hosts to one large, non-blocking switch. In this topology, the throughput of each host is only limited by the capacity of the link that connects it to the switch. Building the optimal topology is not possible in practice, but it provides a useful benchmark against which to compare. Because we assume that all switches are non-blocking, it is not possible to oversubscribe the optimal topology.

5.2 Topology Comparison

The primary focus of this paper is not to compare data center topologies, but since one of PAST's strengths is that can be implemented efficiently on arbitrary topologies, we compared topologies as a matter of course. This section presents the results of that comparison, after which we present a more in-depth evaluation of our

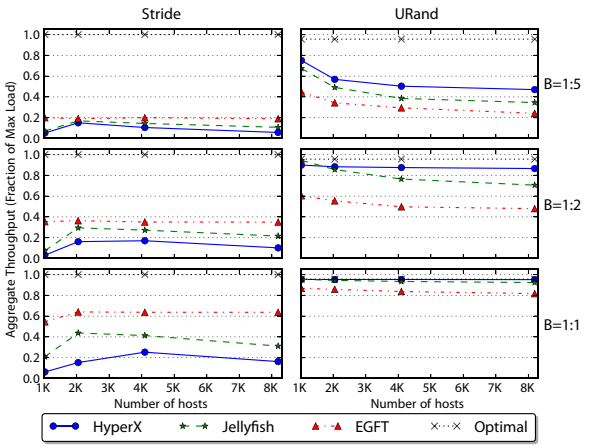

Figure 2: Throughput Comparison of Equal Bisection Bandwidth HyperX, Jellyfish, and EGFT Topologies for Both the Stride-64 and URand-8 Workloads

PAST variants and how they compare to existing data center architectures.

Current best practices for data center networking recommend a Fat Tree or a variant thereof, e.g., EGFT. PortLand [33] describes a routing algorithm for EGFT topologies that scales to arbitrary network sizes and achieves near-optimal performance. If the EGFT topology dominated alternate topologies, there would be little need for better architectures. However, prior work [2, 31, 38] has shown that other topologies can offer equivalent or better performance than EGFTs at lower cost for many workloads. We confirm those observations here showing that Jellyfish [38] and HyperX [2] do provide higher performance at the same cost. We also present the first comparison of the Jellyfish and HyperX topologies, finding that neither dominates the other.

5.2.1 *Performance*

We compare all four topologies assuming ECMP routing because it is the current best practice to make effective use of all available bandwidth and it is the only per-flow multipath mechanism implemented in commodity Ethernet switch hardware. We simulated each of the four topologies for the Stride-64 and URand-8 workloads to compute their aggregate network throughput with ECMP enabled. Figure 2 presents the results of these experiments at three different bisection bandwidth ratios, 1 : 5, 1 : 2, and 1 : 1. Aggregate throughput is normalized to the maximum network load, i.e., when every server transmits and receives at full line-rate. Results for other Stride and URand workloads are omitted for space due to their similarity.

For both workloads, the Jellyfish topology performs somewhere between the HyperX and EGFT topologies. Intuitively, this occurs because the Jellyfish is not optimized for either adversarial, e.g., Stride, or benign, e.g., uniform, traffic, while EGFT is optimized for arbitrary workloads and HyperX is optimized for uniform traffic. This intuition also explains why HyperX performs worst under the adversarial Stride workload. Jellyfish performs better because it is expected to have more minimal paths to nearby hosts than HyperX, and EGFT performs best because minimal routing is optimal on an EGFT.

Under the URand workload, the Jellyfish throughput matches that of the HyperX at low network sizes and gradually decreases as the network size increases. This performance difference is because Jellyfish uses fewer switches and links than a similar size HyperX topology. EGFT underperforms on this workload because it is the most adversely affected by ECMP hash collisions.

		Topology		
Ratio	**Hosts**	**EGFT**	**HyperX**	**JFish**
1 : 1	1K	48	64	57
	2K	96	162	114
	4K	320	338	228
	8K	640	676	456
1 : 2	1K	36	32	38
	2K	70	92	76
	4K	205	210	152
	8K	410	420	304
1 : 5	1K	24	27	26
	2K	50	57	52
	4K	125	119	103
	8K	243	238	206

Table 3: Number of 64-port switches needed to implement each topology.

5.2.2 *Cost*

The previous section compared equal bisection bandwidth topologies, but the different topologies require different numbers of switches to achieve the same bisection bandwidth, and thus have different costs. To account for this, we compare the topologies using a switch-based cost model. While simply counting the number of switches required to implement a given topology does not fully account for cost, it provides a reasonable proxy to compare topologies.

For a fair cost comparison, we implement each topology with the fewest number of switches that provides the necessary bisection bandwidth, subject to host count and switch radix (64-port) constraints. We generated the EGFT and HyperX topologies by searching the space of possible topologies to find the smallest switch count that satisfies the constraints. No search was needed for Jellyfish, because the number of switches in a Jellyfish topology is a function of the network size, bisection bandwidth, and switch radix.

Table 3 shows the number of switches needed for the different topologies. As the network size increases, the number of switches needed for equal bisection bandwidth EGFT and HyperX topologies approach one another. The number of switches needed for the Jellyfish topology remains lower than a comparable size EGFT or HyperX network.

The results in Section 5.2.1 combined with Table 3 show that the performance-to-cost ratio of EGFT is half that of HyperX and Jellyfish for uniform random workloads. Comparing HyperX and Jellyfish is more nuanced. While the performance of the Jellyfish decreases relative to the performance of the HyperX, so does the number of switches used. Also, Jellyfish's flexibility allows a network operator to reverse this trend by adding additional "interior" switches to increase bisection bandwidth. When the performance-per-cost ratios are compared, the results are similar.

5.3 PAST Variants Comparison

In Section 3, we presented two basic PAST routing algorithms, a min-hop destination-rooted variant (PAST) and a non-min-hop one that first routed to an intermediate switch before being forwarded to the final destination (NM-PAST). We also described two ways to

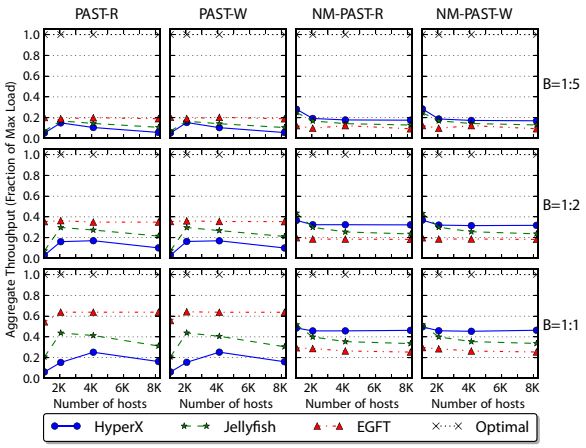

Figure 3: Throughput Comparison of PAST Variants for the Stride-64 Workload

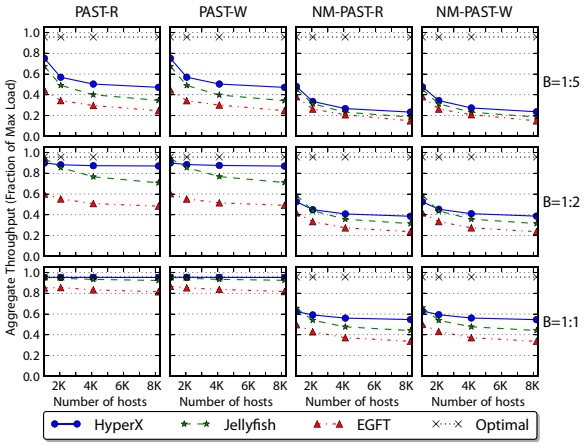

Figure 4: Throughput Comparison of PAST Variants for the URand-8 Workload

build each spanning tree, one that simply chose any random next-hop link (Random) and one that employed a weighted randomization factor to account for non-uniformity in load (Weighted). In total this results in four PAST forwarding mechanisms: *PAST-R*, *PAST-W*, *NM-PAST-R*, and *NM-PAST-W*. We simulated each variant to determine their aggregate throughput at several levels of oversubscription ($1:1$, $1:2$, and$1:5$). The results of these experiments are presented in Figure 3 and Figure 4 for the Stride and URand workloads, respectively.

These results show that both the random and weighted spanning tree algorithms perform identically. Although we omit the results, the different spanning tree algorithms also perform identically on the Shuffle and MSR workloads. One possible explanation for this result is that the regularity of the topologies that we evaluated leads to there being little difference between the spanning trees built by each algorithm. This explanation is corroborated by simulations not presented here on HyperX topologies where different dimensions have significantly different bisection bandwidths. In these simulations, the weighted spanning tree algorithm achieved

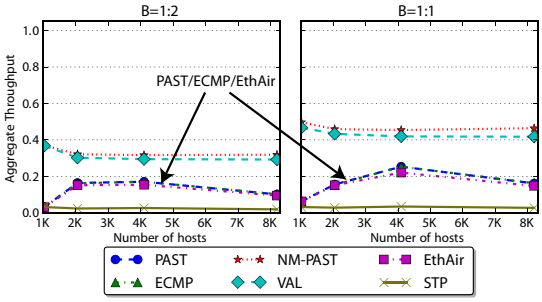

Figure 5: Throughput Comparison of Routing Algorithms for the Stride-64 Workload and HyperX Topology

roughly 10% higher throughput than the uniform random algorithm.

The results for the Stride workload shown in Figure 3 indicate that the NM-PAST algorithm performs better than the PAST algorithm on HyperX topologies. This result is expected because minimally routing the Stride workload on a HyperX causes all flows for a switch to traverse the same small number of links, while Valiant routing takes advantage of the high uniform throughput of a HyperX. In contrast, the PAST algorithm performs better than the NM-PAST algorithm on an EGFT topology. This result is expected because minimal routing on an EGFT already implements Valiant routing, so forwarding to an additional intermediate switch in Valiant routing simply wastes bandwidth.

Surprisingly, under the stride workload, the PAST and NM-PAST algorithms perform almost identically on Jellyfish topologies, even though the routes in NM-PAST are roughly twice the length of routes in PAST. This result arises because there are two competing factors that affect throughput: path diversity and path length. The NM-PAST algorithm increases path diversity at the cost of increasing path length. Increasing path length causes added contention in the network interior, which reduces available bisection bandwidth. These results show that, for the Stride workload, the performance benefit of the improved path diversity of NM-PAST is cancelled out by the increased path length that it requires.

The results for the URand workload (Figure 4) show that the NM-PAST algorithm achieves half of the throughput of the PAST algorithm. This is because NM-PAST does not increase path diversity on the URand workload, yet effectively doubles the load on the network by increasing the average path length by a factor of 2.

5.4 Routing Comparison

The previous section compared the PAST variants against one another. In this section, we compare the best PAST variants against other scalable routing algorithms. Specifically, we compare weighted PAST and NM-PAST against ECMP, Valiant (*Val*), Ethernet on AIR (*EthAir*), and STP routing algorithms. The Valiant routing algorithm is considered, despite not being a scalable routing algorithm, because it provides a useful comparison for NM-PAST. We assume that ECMP is used in *EthAir*. The STP algorithm is used to show the performance of traditional Ethernet.

Figure 5, Figure 6, and Figure 7 present the throughput of each of the routing algorithms on the Stride, URand, and Shuffle workloads, respectively. Results from the MSR workload are omitted for space reasons because the relatively light load offered from the MSR workload does not induce significant performance differences between most of the routing algorithms. The figures only show re-

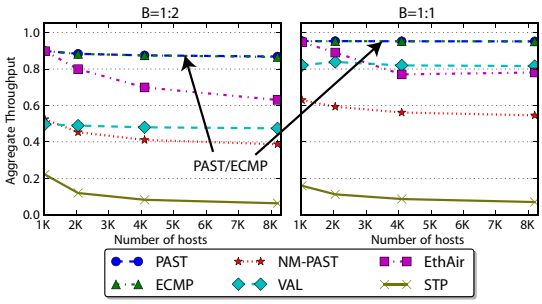

Figure 6: Throughput Comparison of Routing Algorithms Variants for the URand-8 Workload and HyperX Topology

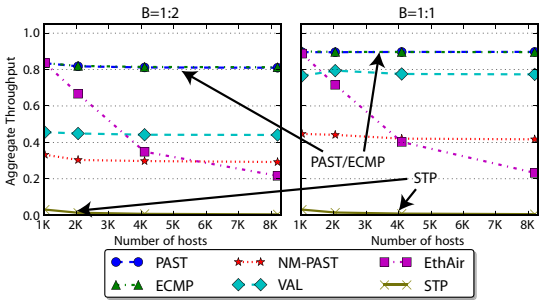

Figure 7: Throughput Comparison of Routing Algorithms Variants for the Shuffle Workload and HyperX topology

sults for the HyperX topology and 1 : 2 and 1 : 1 oversubscription ratio to save space—the other topologies had similar trends.

Our first observation is that the PAST and ECMP routing algorithms perform identically under all workloads. Also, both algorithms perform within 10% of optimal throughput on the 1 : 2 oversubscription ratio topologies for the URand and Shuffle workloads.

NM-PAST uses some minimal paths and does not choose a new random intermediate switch for each flow, so we expect the performance of VAL to be greater than that of NM-PAST. This is the case for the URand and shuffle workloads, but as seen in Figure 5, the NM-PAST and VAL routing algorithms perform similarly on the Stride workload. They are the best performing algorithms, even though the throughput of VAL is far from the optimal throughput, which is equal to the oversubscription ratio [24]. The Stride workload only has a single flow per host, which can cause hash collisions. As the number of flows increase, we anticipate that the throughput of both NM-PAST and VAL will increase.

Overall, the performance of EthAIR is poor. Figure 6 shows that, under the URand workload, the performance of EthAIR is 17%-48% worse than ECMP and PAST. Similarly, under the more demanding shuffle workload shown in Figure 7, EthAIR performs 71%-92% worse than ECMP and PAST at the largest topology size.

The STP algorithm is presented as a strawman to show the baseline performance of traditional Ethernet, even if all broadcasts are disallowed. STP performs significantly worse than all of the other routing algorithms on every topology and bisection bandwidth. This is expected because restricting forwarding to a single tree forces flows to collide.

Table 4 shows the scalability in terms of the number of physical hosts for each of the routing algorithms described earlier, if the routing algorithm were implemented using a network of Trident-based switches. The wildcard ECMP algorithm includes both wildcard ECMP routing on an EGFT topology as well as using EthAir to perform wildcard ECMP on arbitrary topologies. Although both algorithms scale to arbitrary network sizes, the EthAir algorithm restricts the set of usable paths in the network, which reduces performance. Both PAST and STP only require one Ethernet table entry per routable address, so the scalability of both PAST and STP is only limited to the size of the Ethernet table. TRILL ECMP does not scale to networks as large as can be supported by PAST and STP because TRILL ECMP requires ECMP table state, which is exhausted. The scalability of TRILL ECMP is a range because the required number of ECMP table entries varies across topologies and bisection bandwidths.

Wildcard ECMP	PAST	STP	TRILL ECMP
∞	$\sim$100K	$\sim$100K	$\sim$12K-55K

Table 4: Maximum Number of Physical Hosts for Different Eager Routing Algorithms Implemented with Broadcom Trident Chip

6. RELATED WORK

In this section, we discuss the design of PAST in the context of related network architectures. To save space, we omit architectures that have already been discussed, including PortLand, SEATTLE, and Ethernet on AIR. For the sake of discussion, we group related architectures together by the following properties: spanning tree algorithms, link-state routing protocols, and SDN architectures.

MSTP [18], SPAIN [30], and GOE [19] are all architectures that build a spanning tree per VLAN. None of them meet our requirements. MSTP does not achieve high performance because all traffic for a given VLAN is still restricted to a single spanning tree. SPAIN solves this problem by modifying hosts to load balance across VLANs, but SPAIN violates layering and does not scale because each host requires an Ethernet table entry per VLAN. GOE assigns each switch a VLAN and uses MSTP to build a unique spanning tree for each switch. This design limits the total network size to roughly 2K switches and decreases available path diversity and performance compared to PAST. Additionally, all of these architectures limit performance and scalability by requiring broadcast for address learning.

TRILL [20, 36] and Shortest-Path Bridging (SPB) [10] both use IS-IS link-state routing instead of the traditional spanning tree protocol. IS-IS may either use single path or multipath routing. Single path routing limits TRILL to forwarding on switch addresses instead of host addresses, and in SPB it restricts the number of forwarding trees. Using ECMP for multipath routing improves the performance of both architectures, but limits their scalability to the size of the ECMP table. As a result, PAST is more scalable. TRILL and SPB both require specific hardware support that is present in some but not all commodity switch chips, while PAST is designed to use the same hardware as classic Ethernet.

Hedera [4] and DevoFlow [7] are SDN architectures that provide additional functionality compared to PAST. Hedera, DevoFlow, and PAST all eliminate broadcasts and eagerly install routes, but Hedera and DevoFlow both improve performance by explicitly scheduling large flows onto better paths. PAST can complement these traffic engineering mechanisms by efficiently routing flows that are too small to merit explicit scheduling (non-elephant flows). We plan to explore traffic engineering in conjunction with PAST, which will benefit from the fact that PAST does not use the TCAM, so all TCAM entries can be used for traffic engineering.

7. CONCLUSIONS AND FUTURE WORK

Data center network designs are migrating from low-bisection-bandwidth single-rooted trees with hybrid Ethernet/IP forwarding to more sophisticated topologies that provide substantial performance benefits through multipathing. Unfortunately, existing Ethernet switches cannot efficiently route on multipathed networks, so many researchers have proposed using programmable switches (e.g., with OpenFlow) to implement high-performance routing and forwarding. Unfortunately, most OpenFlow firmware implementations and other architectures do not exploit the full capabilities of modern Ethernet switch chips.

In this paper, we presented PAST, a flat layer-2 data center network architecture that supports full host mobility, high end-to-end bandwidth, self-configuration, and tens of thousands of hosts using Ethernet switches built from commodity switch chips. We demonstrate that by designing a network architecture with explicit consideration for switch functionality—in particular the exact-match Ethernet table—it is possible to support heavily multipathed topologies that allow cost and performance tradeoffs. We show that PAST is able to provide near-optimal throughput without using a Fat Tree network. We further show that it is possible to perform efficient multipath routing without using ECMP or similar hashing hardware, which simplifies route computation and installation and could reduce hardware complexity because using ECMP is guaranteed to require more hardware than PAST. Finally, we show that PAST can be easily extended to provide non-shortest-path routing, which benefits adversarial workloads. In the worst case, PAST performs the same as ECMP, while in the best case PAST more than doubles the performance of ECMP.

PAST has implications for the design of future Ethernet switch chips, since our results indicate that layer-2 ECMP is not as useful (or necessary) as previously assumed. We believe PAST will scale well with future networks because the SRAM-based Ethernet table is area-efficient and can easily be increased in size, while it is costly to increase TCAM table size.

Although much has been written about network topologies, we have presented the first three-way comparison between EGFT, HyperX, and Jellyfish. We have also evaluated oversubscribed networks, revealing that in some cases they provide very similar performance to full-bisection-bandwidth topologies but at lower cost. In general, we agree with previous work that Fat Trees are not ideal for any use case. Our work does not provide insight regarding whether HyperX or Jellyfish is the better topology; the outcome is likely to depend on practical considerations, such as ease of cabling, and thus may vary between data centers.

We are excited by the potential of PAST for supporting large enterprise and cloud data centers. We plan to extend it in a number of ways. For example, we are working on an online variant of the per-address spanning tree algorithm that attempts to minimize the amount of new state that needs to be computed and installed when the physical topology or set of addressable hosts changes. We also plan to develop a more detailed cost model for comparing equal-performance HyperX and Jellyfish topologies. Finally, we are exploring ways to integrate traffic engineering, traffic steering, converged storage, high availability, and other advanced networking features into our PAST architecture.

8. ACKNOWLEDGEMENTS

We thank our shepherds, Andrew Moore and Chuanxiong Guo, and the anonymous reviewers for their comments. We also thank Joe Tardo and Rochan Sankar from Broadcom for providing detailed information about the Trident architecture and permission to publish some details here.

References

[1] D. Abts and J. Kim. *High Performance Datacenter Networks: Architectures, Algorithms, and Opportunities*. Morgan and Claypool, 2011.

[2] J. H. Ahn, N. Binkert, A. Davis, M. McLaren, and R. S. Schreiber. Hyperx: topology, routing, and packaging of efficient large-scale networks. *SC Conference*, 2009.

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

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

[5] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan. Data center TCP (DCTCP). In *SIGCOMM*, 2010.

[6] Broadcom BCM56846 StrataXGS 10/40 GbE Switch. http://www.broadcom.com/products/features/BCM56846.php.

[7] A. R. Curtis, J. C. Mogul, J. Tourrilhes, and P. Yalagandula. DevoFlow: Scaling flow management for high-performance networks. In *SIGCOMM*, 2011.

[8] W. Dally and B. Towles. *Principles and Practices of Interconnection Networks*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2003.

[9] N. Farrington, E. Rubow, and A. Vahdat. Data center switch architecture in the age of merchant silicon. In *Hot Interconnects*, 2009.

[10] D. Fedyk, P. Ashwood-Smith, D. Allan, A. Bragg, and P. Unbehagen. IS-IS Extensions Supporting IEEE 802.1aq Shortest Path Bridging. RFC 6329, Apr 2012.

[11] Floodlight openflow controller. http://floodlight.openflowhub.org/.

[12] I. Gashinsky. SDN in warehouse scale datacenter v2.0. In *Open Networking Summit*, 2012.

[13] A. Greenberg, J. Hamilton, D. A. Maltz, and P. Patel. The cost of a cloud: Research problems in data center networks. In *ACM CCR*, January 2009.

[14] A. Greenberg, J. R. Hamilton, 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 *SIGCOMM*, 2009.

[15] Greg Linden. Make data useful. http://www.scribd.com/doc/4970486/Make-Data-Useful-by-Greg-Linden-Amazoncom, 2006.

[16] C. Guo, G. Lu, H. J. Wang, S. Yang, C. Kong, P. Sun, W. Wu, and Y. Zhang. SecondNet: A data center network virtualization architecture with bandwidth guarantees. In *Co-NEXT*, 2010.

[17] IBM BNT RackSwitch G8264. http://www.redbooks.ibm.com/abstracts/tips0815.html.

[18] IEEE. *Std 802.1s Multiple Spanning Trees*. 2002.

[19] A. Iwata, Y. Hidaka, M. Umayabashi, N. Enomoto, and A. Arutaki. Global Open Ethernet (GOE) system and its performance evaluation. *Selected Areas in Communications, IEEE Journal on*, 2004.

[20] J. Touch and R. Perlman. Transparent Interconnection of Lots of Links (TRILL): Problem and Applicability Statement. RFC 5556, May 2009.

[21] S. Jain, Y. Chen, Z.-L. Zhang, and S. Jain. Viro: A scalable, robust and namespace independent virtual id routing for future networks. In *INFOCOMM*, 2011.

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

[23] C. Kim, M. Caesar, and J. Rexford. Floodless in SEATTLE: A scalable Ethernet architecture for large enterprises. In *Proceedings of ACM SIGCOMM*, 2008.

[24] J. Kim and W. J. Dally. Flattened butterfly: A cost-efficient topology for high-radix networks. In *ISCA*, 2007.

[25] G. Lu, C. Guo, Y. Li, Z. Zhou, T. Yuan, H. Wu, Y. Xiong, R. Gao, and Y. Zhang. ServerSwitch: A programmable and high performance platform for data center networks. In *NSDI*, 2011.

[26] G.-H. Lu, S. Jain, S. Chen, and Z.-L. Zhang. Virtual id routing: A scalable routing framework with support for mobility and routing efficiency. In *MobiArch*, 2008.

[27] M. Mahalingam, D. Dutt, K. Duda, P. Agarwal, L. Kreeger, T. Sridhar, M. Bursell, and C. Wright. VXLAN: A Framework for Overlaying Virtualized Layer 2 Networks over Layer 3 Networks. Internet-Draft draft-mahalingam-dutt-dcops-vxlan-00.txt, IETF Secretariat, Jan. 2012.

[28] A. M. Malcolm Scott and J. Crowcroft. Addressing the scalability of Ethernet with MOOSE. In *DC-CAVES*, 2009.

[29] MC-LAG. http://en.wikipedia.org/wiki/MC_LAG.

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

[31] J. Mudigonda, P. Yalagandula, and J. C. Mogul. Taming the flying cable monster: a topology design and optimization framework for data-center networks. In *USENIXATC*, 2011.

[32] J. Mudigonda, P. Yalagandula, J. C. Mogul, B. Stiekes, and Y. Pouffary. NetLord: a scalable multi-tenant network architecture for virtualized datacenters. In *SIGCOMM*, pages 62–73, 2011.

[33] 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 *SIGCOMM*, 2009.

[34] OFlops. http://www.openflow.org/wk/index.php/Oflops.

[35] S. Ohring, M. Ibel, S. Das, and M. Kumar. On generalized fat trees. *Parallel Processing Symposium, International*, 0:37, 1995.

[36] R. Perlman. Rbridges: Transparent routing. In *INFOCOMM*, 2004.

[37] D. Sampath, S. Agarwal, and J. Gacia-Luna-Aceves. 'ethernet on air' : Scalable routing in very large ethernet-based networks. In *ICDCS*, 2010.

[38] A. Singla, C.-Y. Hong, L. Popa, and P. B. Godfrey. Jellyfish: Networking data centers randomly. In *NSDI*, April 2012.

[39] D. Wischik, C. Raiciu, A. Greenhalgh, and M. Handley. Design, implementation and evaluation of congestion control for multipath TCP. In *NSDI*, 2011.