Towards a Flexible Data Center Fabric with Source Routing

Sangeetha Abdu Jyothi, Mo Dong, P. Brighten Godfrey
University of Illinois at Urbana–Champaign

ABSTRACT

An emerging architecture for software-defined data centers and WANs is the network fabric, where complex application-sensitive functions are factored out, leaving the network itself to provide a simple, robust high-performance data delivery abstraction. This requires performing route optimization, in real time and across a diverse choice of paths. A large variety of techniques have been proposed to provide path diversity for network fabrics. But, running up against the constraint of forwarding table size, these proposals are topology-dependent, complex, and still only provide limited path choice which (we show) can impact performance.

We propose a simple approach to realize the vision of a flexible, high-performance fabric: the network should expose every possible path, allowing a controller or edge device maximum choice. To this end, we observe that source routing can be encoded and processed compactly into a single field, even in large networks, with OpenFlow 1.3. We show that, in addition to the expected decrease in required forwarding table size, source routing supports optimal throughput performance, in some cases significantly higher than some past proposals. We thus believe source routing offers a clean abstraction and efficient implementation for future network fabrics.

Categories and Subject Descriptors

C.2.1 [**Network Architecture and Design**]: Centralized networks

Keywords

Source routing; data centers

1. INTRODUCTION

Data center network architecture is moving towards a network fabric abstraction: the core of the network only provides simple forwarding functionality and complex functions are delegated to the edge. This decoupling of forwarding from other network functions provides us with the opportunity to rethink the design of fabric towards a more flexible and efficient forwarding core [1].

How does the fabric achieve high performance? The key is traffic engineering: the network controller or distributed agents must select paths and load-balance between them to adapt to dynamic traffic patterns. This is a difficult technical problem due to the scale, dynamics, and high performance requirements of modern data centers. Since modifying network-wide switch forwarding entries is slow [2], the most common approach is two-stage: first a set of paths are proactively encoded into the data plane, and then the "edge" of the network load-balances among these in real time. Here the edge may be a server, a hypervisor, or a top-of-rack (ToR) switch; and the edge may optimize its path choice autonomously or as instructed by a central controller.

Encoding the paths into the data plane turns out to be non-trivial. A key constraint is the limited capacity of switch forwarding tables, ranging from roughly 2,000 (in some commodity OpenFlow gear) to 200,000 (for very simple exact-match MAC tables). To encode a diverse set of paths between each source-destination pair, a range of designs have proliferated [3–11]. These designs are (variously) topology-dependent, utilize large numbers of forwarding table rules, complex, and yet still limit the selection of paths. For example, CONGA [3] allows a sender to specify the path only up to a spine or top-level switch. Planck [4] and Shadow MACs [12] (as we will see) severely limits throughput for certain traffic patterns and topologies. Most recently, XPath [6] performs a complex compression to encode a large number of paths; it can use more than 100k forwarding rules and still is not guaranteed to provide all paths.

The goal of this paper is to make the case that the fabric should provide a simple abstraction: The ability to use any physical path. This enables *flexibility* in the sense that any available resource can be used without constraints. Furthermore, we argue that the *right architecture to achieve a flexible fabric is source routing:* the sender at the edge specifies a switch-level path through the network, and switches simply match on identifiers referring to their outgoing ports. In a sense, this approach takes the fabric design to its purest form: the switch is stripped down to the minimal functionality necessary for flexible high performance.

The most obvious benefit of this design is tiny forwarding tables (so small that it opens the possibility of simpler switch hardware). But, as we will argue later, a source-routed fabric is a broader architectural win: it improves achievable throughput, is more robust to connectivity failures, localizes switch forwarding table configurations so they are not dependent on network-wide topology, and essentially eliminates the need for careful updates to forwarding state in fabric switches [2]. It also may improve monitoring and security via path provenance, and ease data plane verification.

OpenFlow does not support IPv4 (loose or strict) source routing, and this would anyway lead to large headers. Another method is to use a stack of MPLS labels, 4 bytes per hop with pushing, swapping, and popping along the path. We show an even more compact method is possible: us-

SOSR 2015, June 17–18, 2015, Santa Clara, CA, USA.
©2015 ACM. ISBN 978-1-4503-3451-8/15/06...$15.00
DOI: http://dx.doi.org/10.1145/2774993.2775005

ing arbitrary bit masks in OpenFlow 1.3, paths of sufficient length can be encoded in a single header field (with around 8-10 bits per hop). Perhaps surprisingly, we can even avoid any packet header modification in the switching fabric.

We are not the first to suggest source routing techniques for the data center [7, 13]. In contrast to that work, our contributions are to: (1) make the case for source routing as an *effective architecture for a flexible fabric*, (2) describe a compact method for implementing source routing within OpenFlow, and (3) quantify the improvements in forwarding table size and achievable throughput and how they depend on the traffic matrix.

2. BACKGROUND

Several traffic engineering (TE) techniques have been proposed to improve the performance of data centers. In this section, we elaborate on the state-of-the-art data center forwarding techniques and their limitations.

The majority of the schemes have been tailored to Clos networks, commonly used in data centers. In the discussion, we will refer to two families of topology. A **leaf-spine** topology is a two-tier Clos network. The switches in the lower tier are called leaves and those in the upper tier are called spines or core nodes. Each leaf is connected to every spine. A **fat tree** is a multi-tier Clos network. In the construction of [14], which we assume throughout the rest of the paper, a fat tree is a three-tier network which, when built using switches of k ports, has $\frac{5}{4}k^2$ switches and up to $\frac{k^3}{4}$ hosts.

CONGA [3] is a traffic engineering scheme designed for leaf-spine topologies. Each leaf keeps track of congestion on its paths and chooses the least-congested path for a newly arriving flowlet. Tags in the VXLAN header are used to identify the next-hop spine node as well as to carry congestion information. CONGA relies on source routing at the first hop to allow the destination to keep track of the path used. However, spine nodes rely on the destination leaf address in the packet for forwarding and hence, requires a forwarding table which can accommodate the number of leaves in the network. The technique was evaluated on two-level Clos networks in [3], and as the network grows to multiple levels it would be increasingly difficult to encode all paths as the number of possible paths would grow exponentially.

Shadow MACs [12] is a forwarding scheme for traffic engineering which uses multiple spanning trees rooted at each destination. Each spanning tree has an associated "shadow" MAC address which allows shifting between the trees through MAC rewriting. Thus, for each destination address, only a limited number of paths are available. For example, **Planck** [4] uses a shadow MAC scheme with four spanning trees per destination. In addition to the overhead in MAC address rewriting, the limited path availability can impact throughput performance.

XPath [6] relies on compression to fit a large number of paths into switches' forwarding tables. It uses a two-step process involving aggregation of paths into path sets and assignment of IDs to each path set. Several optimizations reduce the computation for structured networks, but computation time for random networks is still very high (several hours to days). Hence, this is a computationally intensive and complex technique for achieving explicit path control in data centers. [15] also attempts to assign IDs to paths to facilitate concise representation. MAC addresses are assigned as IDs to paths such that multiple paths with that share a link can be aggregated.

A few data center TE schemes have used or proposed the use of source routing before. **FastPass** [5] is a novel traffic engineering scheme where a centralized controller decides the path as well as the time slot of transmission for each packet in the network. This TE scheme is restricted to tiered networks which are rearrangeably non-blocking. The successful implementation of this scheme would require an efficient source routing scheme. The paper suggests the use of VLANs, IP-in-IP tunnelling or ECMP spoofing for the implementation of source routing. In **SlickFlow** [7], packet headers contain a source route for a primary path and an alternate path. Since the packet contains next hop information for both paths, packet headers are larger. Moreover, the technique requires changes in the core of the network to handle the new SlickFlow header. **SecondNet** [13] is a data center virtualization architecture which uses port-switching based source routing (PSSR) as the forwarding method, implemented using MPLS. MPLS is one of the implementation options we discuss later.

Although source routing has been proposed for data center networks in the above papers, an analysis of impact of source routing has been absent. While [16] makes an attempt in this direction, the objective of the design was to reduce the controller load and optimize its placement. Our paper contributes a performance analysis of the impact of source routing, proposes compact implementations in OpenFlow, and makes the broad architectural case for source routing as a fabric substrate.

3. SOURCE ROUTING BASED FABRIC

In this section, we first discuss why a source-routed fabric is an architectural win over a traditional IP-based forwarding fabric. Then we discuss possible ways of implementing source routing and route selection in data centers.

3.1 The Case for a Source Routed Fabric

Smaller forwarding tables: In basic source routing, each switch only needs to store one rule for each outgoing port. For ease of implementation, we will propose schemes which need slightly more: one rule per (hop number, outgoing port) pair. But even in that case, the forwarding table grows linearly with the diameter of the network and is otherwise independent of the number of switches. Thus, source routing can support *all* possible paths to all destinations in the network while reducing the number of forwarding entries by several orders of magnitude. In the future, this may enable simpler and cheaper switch hardware.

Higher throughput: Unlike IP-based forwarding fabric which encounters path constraints due to forwarding table size restrictions, source routing can support any valid path in the network. This allows us to utilize the full path diversity of the network to achieve higher throughput.

Nearly-static forwarding tables: In source routing, a forwarding table entry at a node is updated only during addition or deletion of a link directly connected to it. Thus, forwarding table entries have only local dependencies. Hence, forwarding tables are unaffected by failures or addition of nodes in the broader network, leading to reduced error and complexity. In contrast, traditional forwarding has global dependencies.

Faster response to failures: Consider a leaf-spine topology with two spines (S_1, S_2) and three leaves (L_1, L_2, L_3). A standard forwarding scheme would enable forwarding along all shortest paths. Now suppose links (L_1, S_2) and (S_1, L_3) fail. At this point, even though the network is still physically connected, L_1 cannot reach L_3 since non-shortest paths are not available in the forwarding table. Connection is re-established only after re-convergence of the control-plane routing protocol.[1] On the other hand, source routing would easily circumvent the failure by forwarding along the path $L_1 \rightarrow S_1 \rightarrow L_2 \rightarrow S_2 \rightarrow L_3$ without waiting for control plane re-convergence. This provides a win under the assumption (which we believe will be often true) that edge-based reaction using source routing can happen more quickly than global routing protocol re-convergence or reaction by a central controller.

Architectural solution for consistent update: Since source routing supports all feasible paths in the network, switching paths is just a matter of changing a header field at the sender. This essentially solves the consistent update problem of IP forwarding fabrics [2, 17] where special care is needed because switching paths requires state changes at multiple switches in the network.

Path provenance: Source routing provides better monitoring and diagnostic capabilities since each link traversed by the packet can be retrieved from the header. This can be applied to congestion monitoring, and also potentially security. Just as in a traditional IP-based fabric, if a single hardware or software edge switch is compromised, it can send packets anywhere in the network and potentially compromise segmentation between tenants. But in a source-routed fabric, the receiving switch can sanity-check the paths of incoming packets to filter some kinds of attacks. (Denial of service attacks, however, would still be possible.)

Ease of verification: Formal data plane verification [18–22] of network policies is an important aspect of operation in modern data center networks. For IP-based forwarding, continuous modeling of all rules on all switches are needed because they run distributed routing protocols that change over time. However, with source routing, the fabric's packet forwarding is stable and predictable. Hence, verification of encapsulation at edge routers is sufficient.

In this paper, we provide the broad vision of possible benefits of a source-routing fabric but will focus on quantitative evaluation of first two points listed above.

3.2 Possible implementations

We propose several possible implementations of source routing that are feasible in current data centers. We do assume a centralized controller which has a global view of the network and is capable of sending necessary information to the edge (edge router or hypervisor). The implementation involves two major stages—how routes are selected at the edge, and the encoding of the source route. We enumerate multiple techniques for each stage, which have varying implications for switch hardware and traffic engineering scheme.

3.2.1 Source route encoding

In source routing, the entire path is inserted in the packet header. We need a mechanism that allows each intermediate router to read its next hop from the encoding. Depending on the capabilities of switches in the network, one of the following methods can be chosen for source routing in the core of the network. The methods are listed in increasing order of packet overhead.

(a) Bit mask and TTL: OpenFlow 1.3 allows arbitrary bit masks which can match any bits in a given field. These bits need not be adjacent or at the beginning of the field. This feature can be used to reduce the overhead associated with path information in the packet header. If the maximum degree (number of ports) of any switch in the network is 256, 8 bits are sufficient to uniquely identify a next hop. In such a network, a 32-bit field can hold path information for 4-hop paths. If the largest switch has 64 ports, using a single IPv6 address field, we could accommodate 21-hop paths ($\lfloor 128/6 \rfloor$). This allows a very compact representation of the route in the packet header.

At each switch, we need to identify the correct set of bits to be read. We propose to rely on an existing hidden pointer in the packet. Time-To-Live (TTL) is an 8-bit field in IP/MPLS headers that keeps track of the lifespan of a packet in the network. If we know the TTL set by the source, the number of hops traversed by the packet can be easily deduced. Thus, TTL can be used as a pointer which gives the location of the path corresponding to the current hop. As a simple example, if the TTL is set to 255 at the source, the maximum degree in the network is 16 (4 bits) and 3 hop paths are supported, a mask of xxxx1111xxxx can be used when the current TTL is 253. Note that this technique can be used only if every router in the network is guaranteed to perform TTL operations correctly.

(b) Bit mask and pointer: In networks where TTL functions are unavailable, an additional pointer field can be used to locate the correct next-hop information. The pointer field is initialized to 1 and incremented at each hop. In addition to bit-mask match rule, each packet has to be matched against an additional rule which increments the value of the pointer. This can be implemented in OpenFlow with appropriate matching on the combination of the pointer and route fields, with a rewrite action applied to the pointer.

(c) Bit mask on switch IDs: In contrast to all other schemes presented here, it is actually possible to implement source routing without any header field rewrites. To accomplish this, we specify the path as a list of globally-unique switch IDs. A path that passes through a specific switch contains its own ID followed by the next-hop neighbor's ID. Hence the forwarding table contains match rules for each pair of adjacent locations in the path with the first entry being the switch's own ID and the second entry as any of its neighbors' ID. This technique does not require a pointer to the current location because it is implicit in the position of a switch's own ID in the path. Unlike the previous schemes, the number of bits required per tag is not dependent on the largest degree, but on the number of switches in the network.

(d) MPLS-based source routing: In networks that do not support OpenFlow 1.3 features, Multi Protocol Label Switching [23] can be used instead of compact representation and arbitrary bit masks. However, the default stack size of MPLS labels in most routers is three. Although it can be increased to four or five, this also increases header overhead since each label requires 4 bytes, so MPLS is more appropriate in networks with small diameter.

Now let us compare these four schemes in a 4096 switch network with maximum switch degree 256 and network di-

[1]This can be patched by installing detours around the failed link, as adopted in MPLS Fast-Reroute and CONGA [3], but that may generally produce a sub-optimal route.

ameter of 5. Consider that an IPv6 address of size 128 bits can used for encoding the path. With bit mask and TTL, we need $8 \cdot 5 = 40$ bits since each of the 256 outgoing ports can be uniquely identified with an 8 bit identifier. Routing table size is $256 \cdot 5 = 1280$, since we have to match each combination of the 5 fields and 256 ports. With an additional pointer field of optimal length (3 bits), the utilized header bits increases to 43 while the forwarding table size remains the same (1280 entries). With switch-ID based routing, 12 bits are required to represent each of the 4096 switches. Hence, we will use $12 \cdot 5 = 60$ header bits and routing table size is again 1280. With MPLS, a stack of 5 labels requires $24 \cdot 5 = 120$ bits. The routing table size is 256 since the outermost label can map to one of the outgoing links.

3.2.2 *Route selection at the edge*

In a data center with virtual machines running client applications, security concerns preclude insertion of source routes at the client VM. Hence, the route can be injected either at the hypervisor or at the edge router.

(a) Hypervisor: The hypervisor can be programmed to encapsulate packets received from the attached VMs with the appropriate source route as well as decapsulate packets received from the network. The relevant source route can either be directly obtained from the centralized controller or computed at the hypervisor based on network conditions. The network state could be directly obtained from the controller or could be learned through any distributed algorithm that runs across hypervisors in the network. A suitable path computation technique can be adopted depending on the traffic engineering scheme.

(b) Edge router: The source routes may also be inserted at the edge routers in the network. The path computation needs to be done at the centralized controller, which pushes instructions to the edge routers. The routing table at edge routers will be much larger compared to the rest of the network, as they need to accommodate rules for source path header encapsulation and decapsulation.

Note that our proposal in this paper is essentially agnostic to the traffic engineering scheme. We expect that source routing will benefit traffic engineering schemes broadly, by providing more choice in paths—as we will see in the next section.

4. ANALYSIS

In this section, we analyze the impact of source routing and several past traffic engineering schemes with respect to two main metrics: (a) forwarding table size and (b) throughput performance.

4.1 Methodology

The evaluation focuses on two commonly used data center topologies – leaf-spine and fat trees. In order to compare the performance of various schemes, it is imperative to understand the characteristics of the topology.

Understanding the topologies: Leaf-spine topology with L leaves and S spine switches has $L * (L-1)$ source-destination leaf pairs, each of which can use any of the S (shortest) spine paths, leading to a total of $L * (L-1) * S$ paths. The spine switch only has to save information on each of the destination leaves leading to L entries. However, each leaf switch has $(L-1)S$ entries for S paths to each of the other leaves in the traditional shortest paths-based routing scheme.

Topology	Total paths	Forwarding table entries in a leaf switch			
		Traditional	CONGA	PLANCK	Source routing
Leaf-spine (L leaves, S spines)	$L(L-1)S$	$(L-1)S$	$(L-1)S$	$4(L-1)$	$8S$
Fat trees (degree K)	$K^6/16$	$K^4/8$	-	$2K^2$	$8K$
Leaf-spine (256 leaves, 32 spines)	2088960	8160	8160	1020	256
Fat trees (degree 20)	$4 * 10^6$	$2 * 10^4$	-	800	160

Table 1: Forwarding table size comparison

In a **fat tree** constructed with degree K switches, there are $K^2/2$ leaves in the lower layer. The number of leaf pairs is $K^4/4$. Connecting each pair of leaves, we can have $K^2/4$ paths (considering $K/2$ next-hops between each adjacent layers). Hence, the total number of valid paths in the network is $K^6/16$. At each leaf, we can have $K^2/2$ outgoing flows to other leaves each of which can use at most $K^2/4$ paths. Hence, the total number of forwarding table entries will be $K^4/8$.

Traffic matrix for throughput comparison: We use Topobench [24] to evaluate throughput performance of various schemes. Throughput is computed as a solution to a linear program whose objective is to maximize the minimum flow across all demands. We extend the framework to accommodate a constrained set of paths according to each forwarding scheme. We evaluate three realistic traffic matrices (TMs): (a) All-to-all (A2A) TM with a flow between every pair of switches, (b) Random Matching TM with one outgoing and one incoming flow per switch, chosen uniform-randomly and (c) Random matching with non-uniform traffic where the matching between servers is uniform-random, but 10% of the flows have a demand 10× bigger than the rest of the flows. A flow is defined per leaf pair and is considered to be the aggregate traffic between hosts in these leaves.

A traffic pattern similar to all-to-all is generated by certain real-world applications (such as the shuffle phase in MapReduce). Random matching is a more demanding traffic pattern [24] since it has a single large flow exiting each leaf. This is somewhat similar to a situation that could occur if an application is allocated all the machines in two racks and performs large transfers between the two halves of the application (e.g. a shuffle). Non-uniform random matching is a representative of the realistic scenario where a small fraction of flows dominate in size, alongside other smaller demands spread throughout the network.

We compare the performance of traditional routing (all shortest paths through a router saved in its routing table), CONGA, Planck and source routing. For source routing, we assume that IPv6 address is used to carry the source route in a network with switch degree at most 256. Hence, the source routing design used for evaluation supports paths of length up to 16.

4.2 Forwarding table size

Forwarding table size in switches is limited due to high cost and power consumption. A typical size of forwarding tables in high-end data center switches today is 144K en-

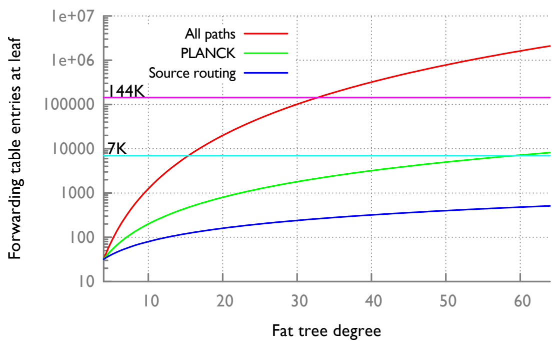

Figure 1: Forwarding table size for fat trees

tries [25], which may not be sufficient to accommodate all paths. The problem is aggravated in low end switches with forwarding table size less than 7K [26]. We compare the requirements of various data center forwarding schemes.

A leaf in the **leaf-spine** topology uses $L \cdot S$ entries under CONGA. Since Planck limits the number of paths to 4 for each destination, the number of forwarding table entries at each leaf is $4L$ in Planck. With source routing, we have a constant number of entries – $8S$. Although we have paths of length 16, a leaf can appear only at an odd position and a spine can appear only at an even position in the path. Hence each switch has to check only half of the locations. This property holds for all "level-based" networks with links only between adjacent levels. A switch at an odd level can appear only at an odd hop in a path.

In a **fat tree topology** with degree K, a leaf switch in Planck has $2K^2$ paths. On the other hand, source routing requires only $8K$ forwarding table entries to support paths up to length 16 in fat tree with shortest path length 5. CONGA does not scale to accommodate three level topologies. Note that it is possible to do hierarchical routing in fat trees with small forwarding tables if control over the chosen path is not required. However, this can lead to congestion and uneven utilization of the network. As we will see, for better utilization of the network and efficient traffic engineering, it is necessary to have information on all paths at the leaves.

A summary of the comparison is given in Table 1. The general trend in the growth of forwarding table size in fat trees with increase in degree is given in Figure 1. We can see that, without any optimizations, forwarding table requirements of traditional routing with all shortest paths cannot be accommodated even in high-end switches for fat trees with degree as small as 34. Planck, with a 4-path limit per destination, cannot support fat trees of degree more than 60, built from low-end switches with $7K$ routing table size. On the other hand, forwarding table requirements of source routing uses only a fraction of the memory in low-end switches. Also, note that the source routing scheme used here can support any paths of length up to 16. Traditional scheme and Planck require forwarding tables which are much larger, while only supporting shortest paths (as shown in the Figure).

4.3 Throughput analysis

Due to the limited forwarding table size, data center switches can accommodate only a limited number of paths at any point in time. This can affect the throughput performance of applications. In this section, we analyze the impact of reducing the number of available paths on maximum achievable throughput. Although several factors can contribute to reduction in throughput, particularly transport mechanisms, this evaluation focuses solely on the limitation imposed by constraints on the available forwarding paths.

We evaluate three realistic traffic matrices (TMs): (a) All-to-all (A2A) (b) Random matching and (c) Random matching with non-uniform traffic. Throughput is normalized with respect to the maximum throughput achievable in the topology using the same TM with no path constraints.

Leaf-spine: We denote a leaf spine topology as (L, S) where L is the number of leaves and S is the number of spine switches. In a leaf-spine topology, a shortest path between leaf switches can be completely specified by a spine switch. We increase the number of paths from 1 to the number of spine switches. For all routing schemes, forwarding table entries increase linearly with the number of paths used. In Figure 2, we plot normalized throughput in the leaf spine topology with 32 spine switches and varying number of leaves under all-to-all TM. We observe that source routing, which can support 16 hop paths, requires the same number of forwarding table entries as traditional routing with only shortest paths (4 hop paths).

Under random matching on the same topology (Figure 3), each new path improves throughput significantly. In other words, limiting the number of paths can impact the throughput significantly when the demand matrix is sparse (unlike the more heavily loaded all-to-all case of the previous plot). Moreover, the number of forwarding table entries required by source routing to achieve maximal throughput is an order of magnitude smaller than that of traditional routing.

Under random matching with non-uniform TM, the plot remains linear. When a fraction of flows have 10× demand, the absolute value of maximum throughput is reduced, but the overall behavior is similar to uniform random matching.

Fat trees: Next, we evaluate the TMs on fat trees, where the number of possible paths grows exponentially with the degree of switches. A fat tree built from degree K switches is denoted by FT-K. Figure 4 shows the variation of normalized throughput in fat trees with all-to-all TM as a function of the number of forwarding table entries at each leaf switch. Under the dense (A2A) TM, 2-3 paths per flow is sufficient for good performance. However, the forwarding table entries grow exponentially with size of the fat tree network.

Figure 5 shows normalized throughput in fat trees under the uniform random matching TM. Limited paths are chosen randomly with additional constraints to use a diverse set of links. In order to minimize overlap between paths, two feasible next hops are picked randomly at each level and the one with fewer paths through it is chosen during the path assignment phase. We observe that the number of forwarding table entries required to maintain the same throughput performance increases dramatically compared to the A2A TM. We also note that performance of Planck with 4 paths per destination degrades as the swich degree (and network size) increases.

Normalized throughput in fat trees with random matching TM and 10% of the flows with 10× bandwidth demand is given in Figure 6. While each flow has equal priority with the uniform traffic matrix, large demand flows are more significant with non-uniform traffic matrix. With minimal overlap between the larger flows, the non-uniform TM can achieve maximal throughput with slightly fewer paths. However, the basic trend remains consistent across the uniform and non-

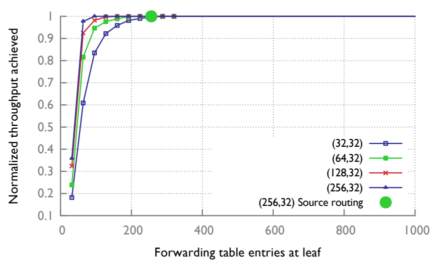
Figure 2: Leaf-spine topology with 32 spine - A2A TM

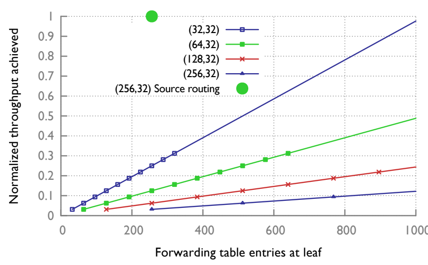
Figure 3: Leaf-spine topology with 32 spine - Random matching

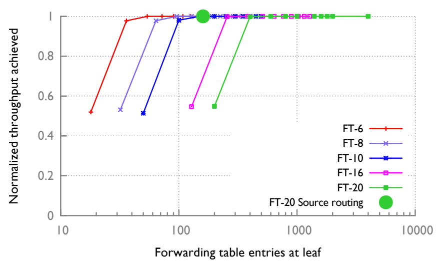
Figure 4: Fat trees A2A TM

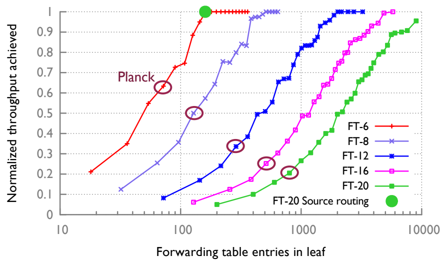
Figure 5: Fat trees Random Matching TM

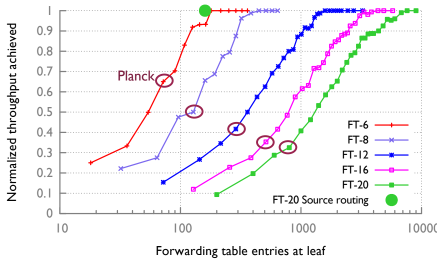
Figure 6: Fat trees Random Matching TM with 10% of flows with 10× demand

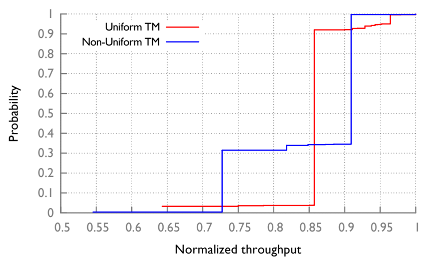
Figure 7: Fat trees Random Matching TM with 10% of flows with 10× demand

uniform random matching TMs since the bottleneck under path constraints is the number of paths itself.

Although the mean throughput exhibits similar trends under uniform and non-uniform TM, the variance properties differ as shown in Figure 7. We conduct 1000 trials on the fat tree constructed from switches of degree 12 with 10 paths allowed per flow, using both uniform and non-uniform random matching TMs. We observe that the uniform TM has a consistent value with very high probability, whereas the non-uniform TM is bimodal. The high variance under non-uniform TMs is due to the randomness in whether the large flows collide. In our experiments, a limited set of paths are pre-assigned to each leaf pair before the flows are assigned. This path assignment has constraints that spread the paths as much as possible. Hence, the paths are nearly uniformly spread for the uniform TM. On the other hand, the distributed path assignment across the network does not guarantee that the large flows are well spread out (since large flows are randomly chosen after all paths are assigned). When the large flows do not collide, the throughput is higher. If the large flows share some links, then the throughput is lower.

We close the evaluation with two take-away points:

Limited forwarding table size will affect throughput performance under worst-case TMs. To achieve high throughput with large rack-to-rack flow groups (i.e. random matching), flows need to have nearly all the possible paths ($K^2/4$) available for forwarding, as seen in Figure 5; and this requirement exceeds the largest forwarding table capacity at switch size of 34 (Figure 1). This illustrates the value of source routing to achieve high throughput in large networks.

A large number of paths between racks can be used effectively. Generally, splitting a flow into a large number of components can affect its performance; $K^2/4$ paths for a flow might seem hard to handle. However, here we are considering aggregate rack-to-rack traffic which is composed of traffic from $(K/2)^2$ host pairs. When a large number of hosts contribute towards the aggregate flow, the number of paths assigned to each host-based flow can be limited. Hence, the paths made available by source routing can be effectively used to improve throughput. Moreover, source routing also allows non-shortest paths which is useful in failure scenarios.

5. CONCLUSION

As data centers move towards network fabric-based architecture, there is an increasing need for a flexible and scalable routing scheme at the core. Under our preliminary investigation, source routing appears to be a technique that fits the bill. With source routing, the forwarding table can easily fit within the available memory of even low-end switches due to its linear dependence on node degree and network diameter. Moreover, throughput performance of the fabric will not be negatively impacted by limited availability in network paths, when source routing allows the edges to pick any feasible path in the network. Due to its flexibility and scalability, source routing is a suitable candidate for network fabric architecture.

In order to strengthen this routing scheme further, we need to tackle a few issues. First, this paper has not dealt with response to switch and link failures. It is essential to build an auxiliary mechanism to route around failures, either with additional information in the header or through a network response mechanism. Second, in order to efficiently utilize the multitude of paths made available by source routing, it is essential to have an efficient traffic engineering scheme. Although a few of the existing schemes rely on several variations of source routing, they are tightly coupled with certain topologies and the performance is far from optimal. With source routing at the core of the design, it will be possible to use any valid path in any topology leading to greater flexibility in the design and implementation of traffic engineering schemes.

We gratefully acknowledge the generous support of National Science Foundation grant CNS 1423452, and Cisco Research Council Grant 580729.

6. REFERENCES

[1] Barath Raghavan, Martín Casado, Teemu Koponen, Sylvia Ratnasamy, Ali Ghodsi, and Scott Shenker. Software-defined internet architecture: Decoupling architecture from infrastructure. In *Proceedings of the 11th ACM Workshop on Hot Topics in Networks*, HotNets-XI, pages 43–48, 2012.

[2] Xin Jin, Hongqiang Harry Liu, Rohan Gandhi, Srikanth Kandula, Ratul Mahajan, Ming Zhang, Jennifer Rexford, and Roger Wattenhofer. Dynamic scheduling of network updates. In *Proceedings of the 2014 ACM Conference on SIGCOMM*, SIGCOMM '14, pages 539–550, 2014.

[3] Mohammad Alizadeh, Tom Edsall, Sarang Dharmapurikar, Ramanan Vaidyanathan, Kevin Chu, Andy Fingerhut, Vinh The Lam, Francis Matus, Rong Pan, Navindra Yadav, and George Varghese. Conga: Distributed congestion-aware load balancing for datacenters. In *Proceedings of the 2014 ACM Conference on SIGCOMM*, SIGCOMM '14, pages 503–514, 2014.

[4] Jeff Rasley, Brent Stephens, Colin Dixon, Eric Rozner, Wes Felter, Kanak Agarwal, John Carter, and Rodrigo Fonseca. Planck: Millisecond-scale monitoring and control for commodity networks. In *Proceedings of the 2014 ACM Conference on SIGCOMM*, SIGCOMM '14, pages 407–418, 2014.

[5] Jonathan Perry, Amy Ousterhout, Hari Balakrishnan, Devavrat Shah, and Hans Fugal. Fastpass: A centralized "zero-queue" datacenter network. In *Proceedings of the 2014 ACM Conference on SIGCOMM*, SIGCOMM '14, pages 307–318, 2014.

[6] Shuihai Hu, Kai Chen, Haitao Wu, Wei Bai, Chang Lan, Hao Wang, Hongze Zhao, and Chuanxiong Guo. Explicit path control in commodity data centers: Design and applications. In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*, May 2015.

[7] R.M. Ramos, M. Martinello, and C. Esteve Rothenberg. Slickflow: Resilient source routing in data center networks unlocked by openflow. In *Local Computer Networks (LCN), 2013 IEEE 38th Conference on*, pages 606–613, Oct 2013.

[8] Radhika Niranjan Mysore, Andreas Pamboris, Nathan Farrington, Nelson Huang, Pardis Miri, Sivasankar Radhakrishnan, Vikram Subramanya, and Amin Vahdat. Portland: A scalable fault-tolerant layer 2 data center network fabric. *SIGCOMM Comput. Commun. Rev.*, 39(4):39–50, August 2009.

[9] Changhoon Kim, Matthew Caesar, and Jennifer Rexford. Seattle: A scalable ethernet architecture for large enterprises. *ACM Trans. Comput. Syst.*, 29(1), February 2011.

[10] Albert Greenberg, James R. Hamilton, Navendu Jain, Srikanth Kandula, Changhoon Kim, Parantap Lahiri, David A. Maltz, Parveen Patel, and Sudipta Sengupta. Vl2: A scalable and flexible data center network. In *Proceedings of the ACM SIGCOMM 2009 Conference on Data Communication*, 2009.

[11] Minlan Yu, Alex Fabrikant, and Jennifer Rexford. Buffalo: Bloom filter forwarding architecture for large organizations. In *Proceedings of the 5th International Conference on Emerging Networking Experiments and Technologies*, CoNEXT '09, pages 313–324, 2009.

[12] Kanak Agarwal, Colin Dixon, Eric Rozner, and John Carter. Shadow MACs: Scalable Label-switching for Commodity Ethernet. In *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*, HotSDN '14, pages 157–162, 2014.

[13] Chuanxiong Guo, Guohan Lu, Helen J. Wang, Shuang Yang, Chao Kong, Peng Sun, Wenfei Wu, and Yongguang Zhang. Secondnet: A data center network virtualization architecture with bandwidth guarantees. In *Proceedings of the 6th International COnference*, Co-NEXT '10. ACM, 2010.

[14] Mohammad Al-Fares, Alexander Loukissas, and Amin Vahdat. A scalable, commodity data center network architecture. In *Proceedings of the ACM SIGCOMM 2008 Conference on Data Communication*, SIGCOMM '08, pages 63–74, 2008.

[15] Arne Schwabe and Holger Karl. Using MAC Addresses As Efficient Routing Labels in Data Centers. In *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*, HotSDN '14, pages 115–120, 2014.

[16] Mourad Soliman, Biswajit Nandy, Ioannis Lambadaris, and Peter Ashwood-Smith. Source routed forwarding with software defined control, considerations and implications. In *Proceedings of the 2012 ACM Conference on CoNEXT Student Workshop*, CoNEXT Student '12, pages 43–44, 2012.

[17] Mark Reitblatt, Nate Foster, Jennifer Rexford, Cole Schlesinger, and David Walker. Abstractions for network update. In *ACM SIGCOMM*. ACM, 2012.

[18] Ehab Al-Shaer and Saeed Al-Haj. FlowChecker: Configuration analysis and verification of federated OpenFlow infrastructures. In *Proceedings of the 3rd ACM workshop on Assurable and usable security configuration*. ACM, 2010.

[19] Haohui Mai, Ahmed Khurshid, Rachit Agarwal, Matthew Caesar, P. Brighten Godfrey, and Samuel T. King. Debugging the data plane with Anteater. In *ACM SIGCOMM*, August 2011.

[20] Peyman Kazemian, George Varghese, and Nick McKeown. Header Space Analysis: Static checking for networks. In *USENIX NSDI*, 2012.

[21] Ahmed Khurshid, Xuan Zou, Wenxuan Zhou, Matthew Caesar, and P. Brighten Godfrey. Veriflow: Verifying network-wide invariants in real time. In *Proceedings of the 10th USENIX Conference on Networked Systems Design and Implementation*, pages 15–28, 2013.

[22] Peyman Kazemian, Michael Chan, Hongyi Zeng, George Varghese, Nick McKeown, and Scott Whyte. Real time network policy checking using header space analysis. In *USENIX NSDI*, 2013.

[23] Framework for Multi-Protocol Label Switching (MPLS)-based recovery, 2003.

[24] Sangeetha Abdu Jyothi, Ankit Singla, Brighten Godfrey, and Alexandra Kolla. Measuring and understanding throughput of network topologies. *CoRR*, abs/1402.2531, 2014.

[25] Arista 7250QX data sheet. `http://www.arista.com/assets/data/pdf/`

Datasheets/7250QX-64_Datasheet.pdf.

[26] Cisco nexus 3000 series data sheet. http://www.cisco.com/c/en/us/products/collateral/switches/nexus-3000-series-switches/white_paper_c11-713535.html.