ProActive Routing in Scalable Data Centers with PARIS

Dushyant Arora
Arista Networks
dushyant@arista.com

Theophilus Benson
Duke University
tbenson@cs.duke.edu

Jennifer Rexford
Princeton University
jrex@cs.princeton.edu

ABSTRACT

Modern data centers must scale to a large number of servers, while offering flexible placement and migration of virtual machines. The traditional approach of connecting layer-two pods through a layer-three core constrains VM placement. More recent "flat" designs are more flexible but have scalability limitations due to flooding/broadcasting or querying directories of VM locations. Rather than reactively learn VM locations, our PARIS architecture has a controller that *pre-positions* IP forwarding entries in the switches. Switches within a pod have complete information about the VMs beneath them, while each core switch maintains complete forwarding state for part of the address space. PARIS offers network designers the flexibility to choose a topology that meets their latency and bandwidth requirements. We evaluate our PARIS prototype built using OpenFlow-compliant switches and NOX controller. Using PARIS we can build a data center network that supports up to 100K servers.

Categories and Subject Descriptors

C.2.0 [**Computer Communication Networks**]; C.2.1 [**Network Architecture and Design**]

General Terms

Design; Performance

Keywords

Data Center Networking; Network Virtualization; Software-Defined Networking

1. INTRODUCTION

Large data centers have thousands of switches that interconnect tens of thousands of servers running hundreds of thousands of virtual machines (VMs). Operating at this large scale puts tremendous pressure on the design of network topologies, routing architectures, and addressing schemes. Recent designs rely on switches to query directory servers to learn how to reach destination VMs, leading to overhead and delays. We advocate a different approach, where a controller pre-positions IP forwarding entries in the network, based on VM addresses and locations.

1.1 Scaling Data-Center Networks

Traditional data centers divide the network into small pods or clusters (running Ethernet protocol), connected by a core (running IP routing protocols). This design offers flexible VM placement and migration within a pod, plus scalability by routing between pods based on IP prefixes. However, intra-pod communication suffers from the inefficiency of routes constructed by spanning tree, and the overheads of flooding (to unknown destinations) and broadcasting (of ARP traffic). In addition, pod boundaries become artificial constraints on VM placement and migration, since VMs in different pods belong to different subnets.

In response, recent designs offer a "one big virtual switch" abstraction, with flat addressing so a VM can run on any server and retain its IP address when it moves. However, flat addressing introduces scalability challenges. Some solutions, like TRILL [1] improve scalability through link-state routing in the core, while still relying on flooding and broadcasting at the perimeter. To avoid these overheads, VL2 [6] and NVP [11] rely on separate directory servers that store the mapping from a VM's address to its location. However, these directories introduce scalability challenges of their own, requiring many servers to handle the load of queries and updates. In addition, edge switches incur delay to query the directory, and overhead to cache the results and encapsulate packets, particularly for the high volume of traffic entering the data center from the Internet.

SEATTLE [10] improves scalability by distributing directory information over the switches, and directing traffic through an intermediate switch on a cache miss at the ingress switch. However, reactive caching can lead to large forwarding tables, unless the traffic matrix is sparse – a reasonable assumption in enterprise networks, but not necessarily in data centers. To reduce forwarding-table size, PortLand [14] rewrites MAC addresses at the network edge, to enable prefix-based aggregation in the core. However, the design is limited to fat-tree topologies. Both SEATTLE and PortLand require modifications to the switches, precluding the use of legacy equipment.

Related work [12] improves scalability by leveraging unique properties of the data center network. However, related work [12] focuses on improving the scalability of multicast groups and does not scalably support VM mobility.

1.2 ProActive Routing In Scalable DCs

In this paper, we present PARIS, a more scalable way to provide "one big virtual switch" in the popular setting of large private clouds (e.g., Facebook, Google), where each VM has a unique IP address. PARIS routes on flat IP addresses, rather than MAC addresses, to leverage legacy layer-three switches that forward packets based on destination IP prefixes. Our design has a controller that pre-positions forwarding state in the network, based on knowledge of each VM's address and location. Rather than encapsulate packets at the network edge, the bottom few levels of switches store a forwarding entry for all VMs beneath them. To avoid a state explosion further up the hierarchy, the controller partitions the forwarding state across the core switches, so each switch maintains fine-grained forwarding state for a portion of the data center's IP address space.

The resulting design offers scalability (to a large number of VMs), flexibility (to place VMs on any servers), and backwards compatibility (to legacy layer-three switches), as well good path diversity and bisection bandwidth.

Roadmap: The next section discusses the main architectural principles underlying our design. Section 3 presents PARIS, including how we achieve high path diversity and low stretch. We present our prototype and evaluation in Section 4. Section 5 discusses multitenancy within PARIS. Section 6 concludes the paper.

2. ARCHITECTURAL PRINCIPLES

In rethinking how to design scalable data-center networks, we identify four main principles:

Flat layer-three network: Having the entire data center form one IP subnet simplifies host and DHCP configuration, and enables seamless VM migration. However, forwarding on MAC addresses introduces scalability challenges, since the address space is large and flat—forcing the use of broadcasting/flooding which has the side effect of each switch learning location of all hosts in the network. In contrast, IP addresses are easily aggregated, with switches forwarding traffic based on the longest-matching prefix.

Proactive installation of forwarding state: Installing forwarding-table entries before the traffic arrives reduces packet latency and avoids the overhead of learning the information reactively. However, injecting flat addresses into routing protocols (e.g., OSPF or IS-IS) leads to large forwarding tables in every switch. Instead, a logically-centralized controller can pre-position the necessary forwarding-table entries in each switch, based on a network-wide view of the topology and the locations and addresses of VMs. This enables much smaller tables.

Complete forwarding information within a pod: Storing a forwarding-table entry for every VM in every switch would not scale. Yet, a switch could easily store the forwarding information for all VMs in the same pod. Today's low-end switches have room for tens of thousands, or even hundreds of thousands of forwarding entries—enough space for thousands of servers each running (say) 32 VMs. Future switches will have larger tables, to accommodate multi-core servers hosting even more VMs. Storing complete information enables short paths and good path diversity within a pod, and default routes up the hierarchy to the rest of the network.

Partitioned forwarding state in the core: Moving up the hierarchical topology, the number of VMs lying "below" a switch grows exponentially. We can no longer store complete forwarding information inside the core-layer switch. So, we divide the data center address space and store full forwarding state for a portion of the IP address space in each core switch. For example, we can divide a /14 address space into four /16 *Virtual Prefixes* [4] and a core switch with a layer-3 table size of 64K can store forwarding information for an entire /16 virtual prefix. Since we treat IP as a flat address within a pod, VMs with IP address within this prefix may be spread across multiple pods.

3. PARIS ARCHITECTURE

In PARIS, a logically-centralized controller proactively installs forwarding-table entries in the switches. We initially assume a private cloud, where all VMs belong to the same institution and have unique IP addresses. First, we discuss how PARIS operates on a range of core topologies. Next, we describe how to reduce stretch and increase path diversity by having fine-grained forwarding entries for popular destinations. Finally, we look at the architecture components and network dynamics.

3.1 No-Stretch Topology

Data center applications have unique and varied traffic requirements. Some applications require high "east-west" bandwidth, while others require low latency. We begin by looking at the No-Stretch topology shown in Figure 1, which achieves the latter. In this topology, each edge switch stores reachability information to all directly-connected VMs, and each aggregation switch stores reachability information to all hosts in its pod. Each aggregation switch must be connected to all core switches in a complete bipartite graph because each core switch stores reachability information to only a subset of the address space. We do not place any restrictions on the topology at the aggregation and edge layer.

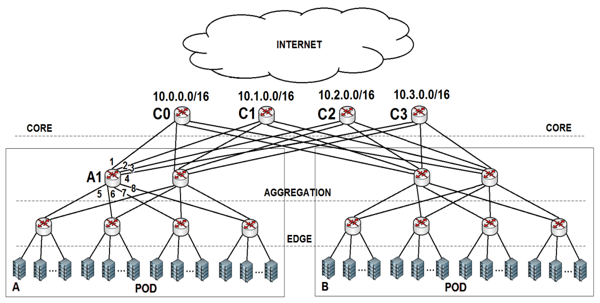


Figure 1: No-Stretch topology with core-layer switches aggregating /16 prefixes

Each *virtual prefix* has an *Appointed Prefix Switch* (APS) in the core layer, responsible for storing forwarding information for all IP addresses that lie within that *virtual prefix*. Aggregation switches store forwarding information for all *virtual prefixes* in order to reach destination VMs in other pods. Figure 1 shows a data center network with 10.0.0.0/14 host address space, where each switch in the core layer is aggregating a /16 IP prefix. We say, for example, that core switch 1 is an APS for virtual prefix 10.1.0.0/16. The actual prefix length depends on the routing table size of the core switches.

Parameterized Construction: Assume a physical server can host V VMs, and let k be the port density of all switches. Since each core-layer switch is connected to all the aggregation-layer switches, we can have maximum k aggregation switches and $2k$ edge switches (for the pod design of Figure 1). The maximum number of hosts supported by this topology will be $2k(k-2) \times V$.

Packets always take the shortest path in the No-Stretch topology. We take advantage of equal-cost multi-path routing (ECMP), wherever possible. In Figure 1, there are two equal-cost paths between every edge and aggregation switch. We represent this as $ECMP_{edge \to agg} = 2$. Traffic between two hosts A (10.0.0.1) and B (10.2.0.24), located in different pods, can take $ECMP_{edge \to agg}$ possible (default) routes from A's edge switch to an aggregation switch inside A's pod. This aggregation switch has forwarding information for all virtual prefixes. It forwards the traffic to the APS aggregating virtual prefix 10.2.0.0/16 in the core layer. The APS can forward the traffic to $ECMP_{edge \to agg}$ number of destination pod aggregation switches. It chooses one using ECMP. The destination pod aggregation switch ultimately forwards the traffic to B via its edge switch.

3.2 High-Bandwidth Topology

In order to support higher bisection bandwidth and make the architecture more scalable, we now look at another flavor of PARIS.

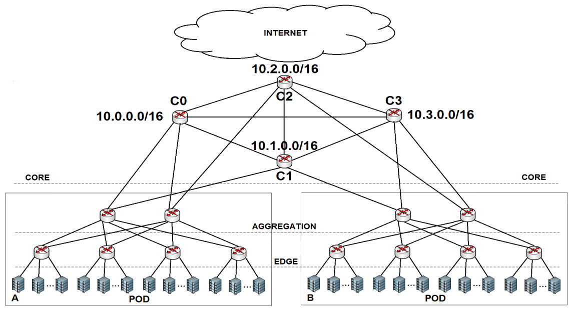


Figure 2: High-Bandwidth topology with a full mesh in the core layer

Instead of having disjoint switches in the core layer, we connect them to form a full mesh (See Figure 2). We choose a full-mesh topology because it has low diameter and multiple paths for fault tolerance and load balancing. Each aggregation switch is connected to $ECMP_{agg \to core}$ number of randomly selected core switches. The aggregation-layer switches use ECMP to spread traffic across these core-layer switches. As before, each core layer switch is an APS. In order to reach all the hosts in the data center, each APS stores reachability information for all virtual prefixes being aggregated by all other APSs in the core layer. The aggregation switches no longer store forwarding information to all virtual prefixes.

Parameterized Construction: Let's say we wish to construct a data center with 2^h hosts using k-port switches, with pod design as shown in Figure 2. The core-layer switches are connected in a complete graph configuration with routing table size 2^r. For connecting these hosts we will need $\lceil \frac{2^h}{(k-2)\times V} \rceil$ edge switches, $\lceil \frac{2^{h-1}}{(k-2)\times V} \rceil$ aggregation switches and 2^{h-r} core switches, where V is the number of VMs each physical server attached to the edge switch can host. Each core layer switch will be connected to $(2^{h-r}-1)$ other core switches. The ECMP value for each aggregation switch will be

$$E = \left\lfloor \frac{(k-2^{h-r}+1)\times(k-2)\times 2V}{2^r} \right\rfloor \quad (1)$$

Each aggregation switch is connected to min(k-4, E) randomly-selected core switches.

Traffic between two hosts A (10.0.0.1) and B (10.2.0.24), located in different pods, flows through a maximum of three core switches (two hops). The source pod aggregation switch hashes the five tuple of the packet and forwards it to one of the core-layer switches (say C0). The ingress core switch looks at the destination IP address of the packet and forwards it to the appropriate APS (C2). The APS then tunnels the traffic to the destination pod aggregation switch through the core layer, if it is not directly connected to it. For tunneling the packets, we can use MPLS labels, VLAN tags, or IP-in-IP encapsulation. In this case, no tunneling is needed and the traffic is directly forwarded to the aggregation switch in B's pod. Traffic between A and B can take $ECMP_{edge \to agg} \times ECMP_{agg \to core}$ number of possible routes from A to the core switch. There are many possible paths through which the packet can reach the destination aggregation switch from the APS. The exact path-multiplicity cannot be determined because of the randomness involved in the link connection procedure used for connecting the aggregation and core-layer switches. The APS and egress core switch use ECMP to spread traffic across the available paths.

3.2.1 *Valiant Load Balancing in Core Layer*

Modern data centers support diverse applications and need to provide good performance through equitable traffic distribution among the links. Measurement studies have found the traffic patterns inside a data center to be highly volatile [6]. *Valiant Load Balancing* (VLB) is an efficient technique for handling traffic variations under the hose model.

In a full-mesh network of N nodes, where a node can send traffic at a maximum rate r to another node, the link capacity of mesh links required to support any traffic matrix using VLB is only $r(\frac{2}{N})$ [17]. For implementing VLB in our topology, the link capacity required between any pair of core switches will be $r(\frac{4}{N})$, where in the first phase the packet is forwarded to an APS after bouncing it off a randomly selected core switch, and in the second phase it is forwarded to an egress core switch after again bouncing it off a random core switch.

By implementing VLB, we can reduce the bandwidth requirement of the internal links in the core layer, and also support arbitrary traffic matrices. However, a packet may now travel four hops instead of two in the core layer, i.e., we trade-off stretch for throughput. This is a reasonable trade-off in data centers, since they have very low network latency.

In conclusion, the High-Bandwidth topology is more scalable, has higher path diversity and can provide higher bisection bandwidth compared to the No-Stretch topology. However, the topology might incur some additional latency in doing so.

3.3 Generalized Core Topologies

So far we have seen No-Stretch and High-Bandwidth variants of PARIS, with varying levels of connectivity between switches in the core layer topologies. These are two special examples of core-layer topologies with different path diversity, stretch, and fault-tolerance properties. We will now look at other graph configurations for connecting the switches in the core layer.

If we have a sparse graph in the core layer with a small number of internal edges, we will have more spare ports for supporting a higher ECMP value at the aggregation layer. But a sparse graph will have a larger diameter and lower path diversity within the core layer. We seek a topology with high connectivity for fault tolerance and load balancing, low diameter, and low vertex degree. These properties are satisfied by *Expander Graphs* which have $O(\log n)$ diameter, where n is the number of vertices in the graph. There are many constructions for different families of expander graphs which have varying vertex degree and diameter. Some of them are LPS graph, Paley graph, Hypercube graph, and superconcentrators. The network designer can choose a suitable expander graph topology depending upon the latency, "east-west" bandwidth, and reliability needs of the data center.

3.4 Fine-Grained Rules for Popular VMs

Traffic to popular destinations may experience stretch in the High-Bandwidth topology, and low bisection bandwidth in No-Stretch topology. To alleviate these problems, the controller can install fine-grained forwarding entries (/32) for popular destinations in the core-layer switches.

In the No-Stretch topology, if the controller installs fine-grained forwarding entries for popular destinations on all core-layer switches, the aggregation switches can use ECMP to spread traffic across the core layer and achieve higher bisection bandwidth for these destinations. For the High-Bandwidth topology, installing individual destination rules on all the core-layer switches, instead of aggregating them at APSs, ensures that all the traffic to these destinations always takes the shortest path through the core layer. For both topologies, the controller needs to install fine-grained forwarding rules only in the core-layer switches.

3.5 Elements and Dynamics

In this section, we discuss the architecture components of PARIS and how they interact. Also, we describe how the network handles external traffic and copes with failure.

3.5.1 Network Elements

Controller/Fabric Manager: The controller has complete visibility into the network topology and knows the address and location of all VMs. Using this initial information the controller performs the following tasks: (i) tracking switch-level topology and host location, (ii) optimally placing and updating forwarding information in switches after startup/failure, and (iii) monitoring network traffic to perform traffic engineering. The controller can be an OpenFlow [13] controller or any entity that can interact with legacy switches through BGP or a command-line interface.

Switches: We do not run any intra-domain routing protocol between the switches. In order to learn about topology changes, the switches must support neighbor discovery via protocols like LLDP and send notification events to the controller. The switches should have the minimal ability to map a destination IP prefix to an outgoing link. Unlike other approaches [5], we don't require the switches to have expensive and power-hungry TCAMs. Our architecture can work on SRAM/DRAM-based switches, which have higher memory density which allows us to aggregate larger virtual prefixes in the APS. Switches should also support ECMP or should be able to have multiple next-hops for the same destination prefix. All the above features are supported by today's commodity switches.

Hosts: Hosts send a variety of broadcast traffic which needs to be managed in order to make the network scalable and save precious bandwidth. We place each host in its own /32 subnet with a default route to its edge switch, so that it no longer sends ARP broadcasts. Unlike other approaches [6, 10, 14], we don't need to lookup directory servers or do special handling of host ARP broadcasts. Host DHCP messages are intercepted by the edge switches and forwarded to the controller, which can assign any unallocated IP address to any host.

3.5.2 External Traffic

Architectures that use packet encapsulation or header rewriting to separate host location and identification [7, 11, 14] must perform these actions on a large volume of diverse traffic arriving from the Internet. For example, Nicira's network-virtualization platform has special gateway servers for this purpose. In contrast, PARIS does not extend or modify the packet header, greatly simplifying the handling of traffic entering the data center from the Internet.

For handling external traffic in the No-Stretch topology, we need a border router which is attached to all the core layer switches and which stores forwarding information for all the virtual prefixes. For the High-Bandwidth topology, we do not need this border router as all the core-layer switches are connected to each other in a full-mesh topology.

3.6 Network Dynamics

Since, the controller has a network-wide view, it plays a crucial role in coping with network dynamics.

Switch Dynamics: If an edge switch fails, the VMs attached to that edge switch become unreachable. Unless there are redundant edge switches, there is nothing that can be done to restore reachability. If an aggregation switch fails, the controller can re-route traffic through other aggregation switches in the pod. If a core-layer switch fails, the virtual prefix being aggregated by it can be sub-divided into smaller sub-prefixes and stored on other core-layer switches until a new core-layer switch comes up. This provides graceful degradation and load balancing properties to the architecture.

Link Dynamics: Since, we have multiple paths between any pair of hosts, simple re-routing of traffic by the controller can restore reachability after a link failure. The High-Bandwidth topology is more resilient to link failures than No-Stretch topology because of higher ECMP value between aggregation and core layer.

Host Dynamics (VM Migration): When a VM migrates to a new pod, the controller installs forwarding information on the new edge and aggregation switches for reaching the migrated VM. Also, it updates the APS entry, so that the APS forwards the packets to the new aggregation switch. Finally, the controller deletes forwarding entries from old edge and aggregation switches. The controller can orchestrate VM migration or it can learn about it through gratuitous ARP from the migrated VM.

4. EVALUATION

In this section, we will evaluate the ability of PARIS to scale to large data centers focusing especially on analyzing the trade-offs between latency and bandwidth.

4.1 Simulation Setup

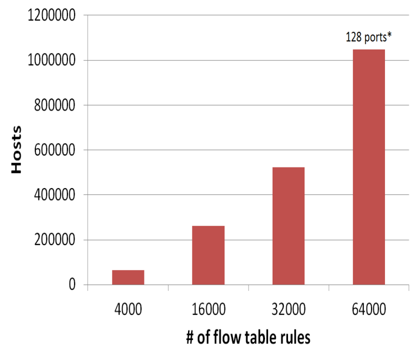
We begin by describing the simulator used to evaluate PARIS as well as the topologies and traffic matrices under which our evaluations are performed.

To evaluate PARIS, we developed an emulator that runs our controller without an actual network. The emulator emulates network events such as switch join events and stores events from the controller such as the flow table entries. This emulator allows us to determine the number of flow table entries installed at each switch as well as determine the runtime of our algorithm under varying network conditions such as link failures.

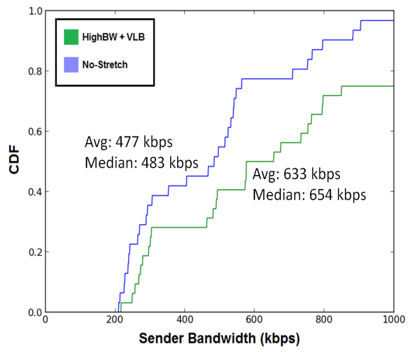
To evaluate the trade off between bandwidth and latency, we use the Mininet-HiFi [9] simulator to simulate a data-center environment. We use OpenFlow [13] switches for constructing the topologies, with NOX [8] serving as their controller/fabric manager. Due to the inherent scalability limitations of the simulator, we are able to perform experiments for moderately small data centers. However, we believe that the high level qualitative difference between No Stretch and High BW remains the same in larger networks.

For our simulations, we evaluate two data center networks, with the same number of hosts (64), edge (32) and aggregation switches (16), but different number of core switches (4 and 8). For a fair comparison, when a network with a given core switch count is configured, it uses the same number of switches in each layer across all topologies. The different number of core switches gives rise to different $ECMP_{agg \rightarrow core}$ values for the High-Bisection topology.

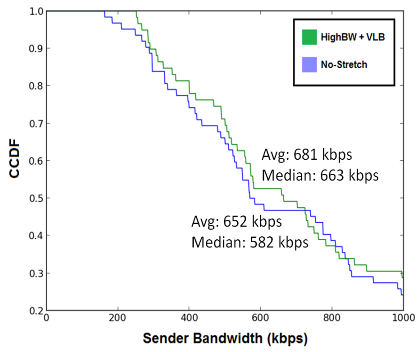
The network has no over-subscription at any layer. We connect the edge switches to the hosts via 1Mbps links and connect the switches to each other using 10Mbps links. The link bandwidths were scaled down compared to those in traditional data centers to make the experiments run faster. For our traffic matrix, we use a



(a)



(b)



(c)

Table 1: (a) Number of table entries at the core switches. (b) CDF of sender bandwidth for No-Stretch and High-Bandwidth PARIS. (c) CCDF of sender bandwidth for No-Stretch and High-Bandwidth PARIS.

random traffic pattern, where each host sends traffic to one randomly selected host for 20s. We use iPerf [2] for generating TCP traffic between the hosts.

4.2 Hierarchal Rule Partitioning: Flow table Scalability

Next, we examine the efficiency of PARIS's rule partitioning algorithms by analyzing its ability of PARIS to scale to large data centers. We evaluate PARIS's ability to minimize and reduce the number of flow table entries stored in core switches.

Figures 1(a) presents the number of flow table entries that each core switch needs to maintain to support data centers of varying sizes. We observe that PARIS is able to effectively support large data centers with as many as 100K physical hosts within the flow table constraints of existing OpenFlow switches (NoviFlow develops switches with as many as 1M flow table entries [15].)

4.3 Trade-offs: Latency versus Bandwidth

Now we examine the throughput and latency of PARIS under various topologies and configurations. First, we examine the latency of routes created by the two modules in PARIS, namely No Stretch and High BW modules. In Table 1(b) we examine the latency for communication between VMs within a pod (intra-pod) and for VMs in different pods (inter-pod).

Topology	Communication Pattern	Average Latency
No-Stretch	intra-pod	61μs
No-Stretch	inter-pod	106μs
High-Bandwidth	intra-pod	61μs
High-Bandwidth	inter-pod	126μs

Table 2: Average latency between VMs.

Our measurements showed that the intra-pod RTT is approximately 61μs for both schemes but the inter-pod RTT for No-Stretch PARIS and High-Bandwidth PARIS is approximately 106μs and 126μs respectively. These results reflect the fact that RTT increases as the average path length increases. Since the packet has to travel a longer path through the core layer in High-Bandwidth PARIS, it has slightly higher inter-pod RTT.

Figure 1(b) shows the CDF of sender bandwidth for both No-Stretch and High-Bandwidth PARIS. Since there is no oversubscription in the network, a sender can at maximum achieve a bandwidth of 1000kbps if there is no flow-collision in the switch layers. As expected, given the random traffic pattern, we see that High-Bandwidth PARIS achieves higher average sender bandwidth compared to No-Stretch PARIS.

We now bias the topology in favor of No-Stretch PARIS and run the experiment again. This time, we create a topology with 64 hosts, 32 edge switches, 16 aggregation switches and 8 core-layer switches. In No-Stretch PARIS, each aggregation switch will be connected to all 8 core switches but in High-Bandwidth PARIS we connect each aggregation switch to 4 random core switches. So, the degree of multi-pathing is reduced. We run the experiment again and we find that High-Bandwidth PARIS still achieves higher average sender bandwidth compared to No-Stretch PARIS. Figure 1(c) shows the complementary CDF for this experiment. We can leverage other solutions like Hedera [3] to make sure flows don't collide in the data center network.

5. DISCUSSION

Multi-Tenancy: Our architecture so far was geared towards private clouds, where all VMs have unique IP addresses. However, it can be easily extended to provide multi-tenancy support, where a tenant can request any (possibly overlapping) IP address. Unlike VL2 [6] and NVP [11], we don't need to tunnel the tenant traffic using the layer-three network as an overlay, or query central directories for encapsulation information.

To uniquely identify each VM in the data center network, we allocate each tenant a unique ID. The edge and aggregation switches store the tenant IDs along with the IP addresses of all VMs beneath them in the pod. The edge switches are responsible for tagging (MPLS/VLAN) the traffic coming from a VM with its tenant ID, and removing the tag from the traffic going to a VM. The APS aggregate virtual IP prefixes as before. However, they now also store tenant IDs along with the /32 IP addresses. So, they might have multiple entries with the same /32 IP address but different tenant IDs. The virtual prefix size now not only depends on the routing table size, but also the number of tenants we wish to support in the data center network. There is a single entry for each tenant VM in the edge and aggregation switches in its pod, and in the APS under whose virtual prefix the tenant IP address lies. Traffic is routed as before, but now using the combination of tenant ID and VM IP address to uniquely identify each VM. We plan to implement our architecture for the multi-tenant setting in future work.

Controller Scalability: In PARIS, the OpenFlow controller handles the following tasks: (1) tracking VM location (migration), (2) tracking the network topology, (3) calculating forwarding entries,

(4) responding to ARP messages, and (5) responding to DHCP messages. While this imposes a non-trivial amount of load on the controller, prior work [16] has shown that by distributing the OpenFlow controller, the controllers can scale to clouds with 100K servers and 2M VMs.

6. CONCLUSION

In this work, we demonstrate how Proactive Routing on flat IP addresses can be used to build scalable and flexible data center networks. We eliminate flooding/broadcasting of packets and avoid querying directories for VM location. Instead, we use a controller to pre-position forwarding state in each switch layer. We propose a new core-layer topology which provides us with increased multi-pathing and bisection bandwidth. We also show how, through installation of fine-grained forwarding entries for popular destinations, we can further improve performance. Finally, we evaluate our architecture on Mininet-HiFi using NOX controller and user-space software OpenFlow switches.

7. REFERENCES

[1] IETF TRILL working group. `http://www.ietf.org/html.charters/trill-charter.html`.

[2] Iperf. `http://iperf.sourceforge.net/`.

[3] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat. Hedera: Dynamic Flow Scheduling for Data Center Networks. In *Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation*, NSDI'10, pages 19–19, Berkeley, CA, USA, 2010. USENIX Association.

[4] H. Ballani, P. Francis, T. Cao, and J. Wang. Making Routers Last Longer with ViAggre. In *Networked Systems Design and Implementation*, 2009.

[5] C. Clos. A study of Non-blocking Switching Networks. *Bell System Technical Journal*, 32:406–424, 1953.

[6] 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 *Proceedings of the ACM SIGCOMM 2009 Conference on Data Communication*, SIGCOMM '09, pages 51–62, New York, NY, USA, 2009. ACM.

[7] A. Greenberg, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta. Towards a Next Generation Data Center Architecture: Scalability and Commoditization. In *Proceedings of the ACM Workshop on Programmable Routers for Extensible Services of Tomorrow*, PRESTO '08, pages 57–62, New York, NY, USA, 2008. ACM.

[8] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker. NOX: Towards an Operating System for Networks. *SIGCOMM Comput. Commun. Rev.*, 38(3):105–110, July 2008.

[9] N. Handigol, B. Heller, V. Jeyakumar, B. Lantz, and N. McKeown. Reproducible Network Experiments Using Container-based Emulation. In *Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies*, CoNEXT '12, pages 253–264, New York, NY, USA, 2012. ACM.

[10] C. Kim, M. Caesar, and J. Rexford. Floodless in SEATTLE: A Scalable Ethernet Architecture for Large Enterprises. In *Proceedings of the ACM SIGCOMM 2008 Conference on Data Communication*, SIGCOMM '08, pages 3–14, New York, NY, USA, 2008. ACM.

[11] T. Koponen, K. Amidon, P. Balland, M. Casado, A. Chanda, B. Fulton, I. Ganichev, J. Gross, N. Gude, P. Ingram, E. Jackson, A. Lambeth, R. Lenglet, S.-H. Li, A. Padmanabhan, J. Pettit, B. Pfaff, R. Ramanathan, S. Shenker, A. Shieh, J. Stribling, P. Thakkar, D. Wendlandt, A. Yip, and R. Zhang. Network virtualization in multi-tenant datacenters. In *Proceedings of the 11th USENIX Conference on Networked Systems Design and Implementation*, NSDI'14, pages 203–216, Berkeley, CA, USA, 2014. USENIX Association.

[12] X. Li and M. J. Freedman. Scaling ip multicast on datacenter topologies. In *Proceedings of the Ninth ACM Conference on Emerging Networking Experiments and Technologies*, CoNEXT '13, pages 61–72, New York, NY, USA, 2013. ACM.

[13] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. OpenFlow: Enabling Innovation in Campus Networks. *SIGCOMM Comput. Commun. Rev.*, 38(2):69–74, Mar. 2008.

[14] R. Niranjan 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 *Proceedings of the ACM SIGCOMM 2009 Conference on Data Communication*, SIGCOMM '09, pages 39–50, New York, NY, USA, 2009. ACM.

[15] NoviFlow. 1248 Datasheet. `http://bit.ly/1baQd0A`.

[16] A. Tavakoli, M. Casado, T. Koponen, and S. Shenker. Applying NOX to the Datacenter. In *ACM SIGCOMM HotNets Workshop*, 2009.

[17] R. Zhang-Shen and N. McKeown. Designing a Predictable Internet Backbone Network. In *ACM SIGCOMM HotNets Workshop*, 2004.