2012 IEEE Fifth International Conference on Cloud Computing

GARDEN: Generic Addressing and Routing for Data Center Networks

Yan Hu[1], Ming Zhu[2], Yong Xia[1], Kai Chen[3], Yanlin Luo[1]
[1] NEC Labs China, [2] Tsinghua University, China, [3] HKUST, Hong Kong
Contact email: yhu.woomy@gmail.com

***Abstract*—Data centers often hold tens to hundreds of thousands of servers in order to offer cloud computing services at scale. Ethernet switching and IP routing have their own advantages and limitations in building data center networks. Recent research, such as PortLand and BCube, has proposed scalable data center network designs. A common feature of these designs is that their addressing and routing are customized to specific topologies.**

In this paper, we propose a generic addressing, routing and forwarding protocol for data center networks, which works on arbitrarily "layered" network topologies. We first form the network as a multi-rooted tree. Each network node (i.e., hosts and switches) is then assigned one or more locators, and each locator encodes a downward path from the roots to this node. Data center networks often have rich path diversity, so tracking all locators of a destination node will cause switches to have very large forwarding tables. We further use a new forwarding model to reduce the forwarding states. In addition, the multiple-locator mechanism brings built-in support for multi-path routing, load balancing and fault tolerance. Evaluations based on simulations and prototype experiments demonstrate that our proposal achieves our design goals.

I. Introduction

Cloud computing services have been driving the creation of data centers that hold thousands to even hundreds of thousands of commodity servers. To make these cloud computing services fast and reliable, it is critical to build cloud data center networks that have high bandwidth and low latency, scale well, are easy to manage and resilient to network failures.

To support a wide variety of cloud computing services, as discussed in [1], a competitive data center network has to satisfy a couple of requirements: a) *low management and operation cost:* minimize configuration and management overhead, facilitate flexible application placement, and provide mobility support for virtual machine migration; b) *traffic efficiency:* fully utilize network link capacity to support high bandwidth and/or low latency applications; c) *scalability:* keep forwarding tables small and introduce little flooding traffic, even if network grows large; d) *fault tolerance:* quickly detect and recover from network failures.

Traditionally, data center networks have been built using Ethernet switches and IP routers. Ethernet excels at zero configuration and easy management, but has suboptimal traffic efficiency and poor scalability. In contrast, IP routers can achieve shortest path routing over arbitrary network topologies and scale better, but its configuration and management overhead are much higher.

There have been much recent research on designing data center networks to meet the goals discussed above. These include VL2 [1], PortLand [2], DCell [3], and BCube [4], etc. In section II, we review these proposals in more details. One common issue of all these proposals is that each of them is targeted to a specific network topology (e.g., clos for VL2, fat-tree for PortLand, and hypercube for BCube). The addressing, routing, and traffic scheduling protocol design of each proposal is thus tightly tied to a specific topology, which limits customers' choice of network topology. To provide, but not dictate, choices to its customers, a network equipment vendor has to implement all these different protocols and cause higher cost.

In this paper, we present a new data center network design that can work on arbitrarily "layered" network topologies. We start from two basic observations for data center networks. First, data center networks, either in the traditional settings [5] or in the new proposed topologies like clos [1] and fat-tree [6], are often physically inter-connected as a hierarchical structure, rather than a random graph. Second, data center networks often have rich path diversity. The topologies are designed to allow multiple paths between any pair of nodes for load-balancing or robustness. Based on the two observations, we designed a set of addressing and routing mechanisms, as detailed in Sections III-V.

Given a data center network topology, each switch or host is assigned one or more locators. Locators encode the node's location in the topology. The first step of addressing is to form the topology as a multi-rooted tree. Then, for each node, we find out all the downward paths from the roots to this node. For each path, we assign a locator to this node. Locators, instead of the MAC address, are used in forwarding. Since locators are addressed according to the hierarchical structure of the multi-rooted tree, address aggregation can reduce the forwarding states, compared to the flat MAC address. A central controller assists with address resolution, path selection and fault tolerance. In address resolution, the controller selects a path between the host pair, and returns the corresponding locator of the destination host to the source host. Multiple locators indicate multiple paths. Various scheduling algorithms can be used in locator/path selection. In addition, network failure can be easily handled

978-0-7695-4755-8/12 $26.00 © 2012 IEEE
DOI 10.1109/CLOUD.2012.9

by replacing the original locator corresponding to the failure path with other locators.

Since a node's locators encode all the paths from the roots to this node, given the fact that data center networks often have rich path diversity, tracking all the locators of all destination will cause switches to have a very large number of forwarding states. A key question is how to reduce the forwarding states while at the same time retain the routing efficiency and multi-path capability. To this end, we use two mechanisms. We first introduce a new forwarding model, such that traffic through a switch can be divided into three forwarding types. The second mechanism is to utilize input port in forwarding, in addition to the destination locator.

In summary, the main contribution in this paper is to propose a set of generic addressing, routing and forwarding mechanisms that are optimized for common data center topologies, and can support all "layered" topologies constructed in arbitrary ways. The forwarding model reduces forwarding states while keeping the routing efficiency and multi-path capability. And the multiple-locator mechanism brings built-in support for multi-path routing, load balancing and fault tolerance. The simulation (Section VI) and prototype experiment (Section VII) results validate these claims.

II. Background and Related Work

In Ethernet switching, the "plug-and-play" semantics offered by persistent MAC addresses and its self learning mechanism greatly simplifies network configuration and management. But Ethernet cannot be directly applied to current mega data center because of its two problems. First, it uses the spanning tree protocol (STP) to ensure a loop-free topology. STP performs well for small network; But, in large network with many redundant paths, STP introduces substantial inefficiency. Second, the network-wide flooding and ARP broadcasting would result in large overhead that grows with network size. There are many proposals to enhance Ethernet switching [7], [8], [9].

IP routing can scale to large networks and ensures efficient usage of networking resources via shortest path routing. However, the tradeoff is the complexity in configuration and management. Routing protocols like OSPF have other limitations such as control traffic flooding, lack of multi-path support and slow converge. In addition, virtual machines (VM) are being widely deployed in data centers. Migrating a VM to a different switch would require assigning a new IP address, an operation that would break all the ongoing TCP connections on the VM.

An ideal data center network architecture should combine the advantages of Ethernet switching and IP routing, that is, not only "plug-and-play", but also scalable, efficient, and robust. Recently, there have been a number of proposals targeting these goals. For example, PortLand assigns Pseudo MAC (PMAC) addresses to all end hosts to encode their position in the topology [2]. Packet forwarding proceeds based on the hierarchical PMAC, which results in very small forwarding tables. However, PortLand is designed for the fat tree topology, but can not be directly used in arbitrary network topologies. VL2 provides a virtual layer 2 network over the clos topology [1]. It separates names (AA) from locators (LA), and uses a directory service to maintain the mapping between them. Switches run the OSPF link state routing protocol and maintain switch level topology. VL2 uses VLB and ECMP to spread traffic over multiple paths.

	Topology	Multipath	FWS	Address	Routing
GARDEN	layered	arbitrary	O(S)	locators	generic
PortLand	fat-tree	ECMP	O(P)	PMAC	specific
VL2	clos	VLB	O(S)	AA/LA	OSPF
SPAIN	arbitrary	arbitrary	O(NV)	flat	VLAN
SEATTLE	arbitrary	single	O(N)	flat	OSPF

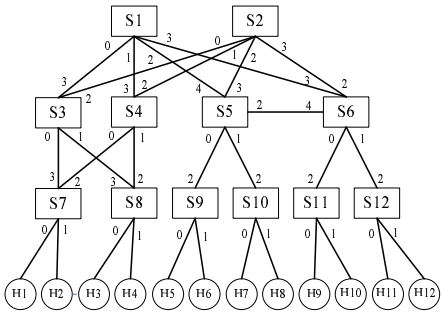
Table I
Comparison of GARDEN and the related work

SPAIN provides multi-path forwarding over arbitrary topologies [10]. It first pre-computes a set of paths to exploit the redundancy in a given network topology, then merges these paths into a set of trees, and each tree is mapped as a separate VLAN. Flat MAC address is used in SPAIN, so the forwarding table size increases proportionally to the total number of hosts in the network. SEATTLE proposes a scalable Ethernet architecture for large enterprises [11]. The switches combine link-state routing and an one-hop, network-wide DHT to achieve broadcast elimination and shortest-path forwarding. We compare our proposal with the above-discussed work in Table I, where FWS indicates the forwarding state, P is the number of ports per switch, S is the number of switches, N is the number of end hosts, and V is the number of VLANs in SPAIN.

DAC presented an automatic data center address configuration system [12]. Rather than solving the addressing and routing problems, their goal is to map a blueprint which contains connectivity information and logical id for servers and switches, to the real physical network and complete the configuration task. ALIAS [13] presented an addressing and communication protocol that automates topology discovery and address assignment for the hierarchical topologies of data centers. ALIAS groups switches into hypernode (HN) to reduce forwarding state. However, topology fluctuations are easy to change the HN membership, and lead to relabeling of switches and hosts.

III. Design Overview

Our design employs a logically centralized controller, which assists with addressing, routing, address resolution, path selection and fault tolerance. After topology discovery, the central controller assigns locators for switches and hosts, and then calculates the forwarding tables for each switch based on the address space.



locators		
S1: 0	S7: 0.0.0 1.0.0 0.1.0 1.1.0	H1: 0.0.0.0 1.0.0.0 0.1.0.0 1.1.0.0
S2: 1		
S3: 0.0 1.0	S8: 0.0.1 1.0.1 0.1.1 1.1.1	H3: 0.0.1.0 1.0.1.0 0.1.1.0 1.1.1.0
S4: 0.1 1.1		
S5: 0.2 1.2	S9: 0.2.0 1.2.0	H5: 0.2.0.0 1.2.0.0
S6: 0.3 1.3		

forwarding tables: dest loc -> port		
S1 0.0.* -> 0 0.1.* -> 1 0.2.* -> 2 0.3.* -> 3	S3 0.0.0.* -> 0 1.0.0.* -> 0 0.0.1.* -> 1 1.0.1.* -> 1 1.* -> 2 0.* -> 3	S7 0.0.0.0.* -> 0 1.0.0.0.* -> 0 0.1.0.0.* -> 0 1.1.0.0.* -> 0 0.0.0.1.* -> 1 1.0.0.1.* -> 1 0.1.0.1.* -> 1 1.1.0.1.* -> 1 0.1.* -> 2 1.1.* -> 2 0.0.* -> 3 1.0.* -> 3 0.* -> 2 or 3 1.*-> 2 or 3
S5 0.2.0.* -> 0 1.2.0.* -> 0 0.2.1.* -> 1 1.2.1.* -> 1 0.3.* -> 2 1.3.* -> 2 1.* -> 3 0.* -> 4	S9 0.2.0.0.* -> 0 1.2.0.0.* -> 0 0.2.0.1.* -> 1 1.2.0.1.* -> 1 0.* -> 2 1.* -> 2	

Figure 1. Addressing and forwarding table examples. The number near each link indicates the port number of the switch.

A. Addressing

Given a network topology, the first step of addressing is to form the topology as a multi-tiered structure or a multi-rooted tree. Here we first define some notations:

- $lev(x)$: the level of node x in the multi-rooted tree
- $P_j(S_i)$: the j^{th} port of switch S_i
- $dir(P)$: the direction of port P, including Upward Ports (*UwP*), Downward Ports (*DwP*) and Peer Ports (*PrP*)
- $loc(x)$: locators of node x

The input of the algorithm is an undirected graph, where switches and hosts are nodes and the links are edges. The output is a multi-rooted tree, each switch and host are assigned a level, and all ports of switches are assigned directions. Assume the height of the tree is H, then from top to bottom each level is defined as level 0 to $H-1$. For switch S_a and S_b, assume $P_m(S_a)$ is connected to $P_n(S_b)$. If $lev(S_a) > lev(S_b)$, then we define $dir(P_m(S_a)) = UwP$ and $dir(P_n(S_b)) = DwP$. And if $lev(S_a) = lev(S_b)$, then we define $dir(P_m(S_a)) = dir(P_n(S_b)) = PrP$.

We first sequentially take each switch S_i as the root, create a spanning tree containing all the end hosts, assume the height of the tree is H_i. Let $H = min(H_i)$. Among all the switches, we select those one with height $H_i = H$ as the root switches Sr, and define $lev(Sr) = 0$. For each port of a root switch $P_j(Sr_i)$, if $P_j(Sr_i)$ is connected to another root switch, then we define $dir(P_j(Sr_i)) = PrP$; otherwise, $dir(P_j(Sr_i)) = DwP$. Next, we assign the level and port direction of other non-root switches level by level. For a switch S_a, if $P_m(S_a)$ is connected to $P_n(S_b)$, and $dir(P_n(S_b)) = DwP$, then $lev(S_a) = lev(S_b) + 1$, and $dir(P_m(S_a)) = UwP$. If $lev(S_a) = lev(S_b)$, then $dir(P_m(S_a)) = PrP$. Otherwise, $dir(P_m(S_a)) = DwP$.

After the network topology is formed as a multi-rooted tree, all the switches and hosts are assigned one or more locators. Locators encode the node's location in the topology, specifically, they encode all paths from the root switches to this node. Firstly, the R root switches are assigned locators: "0", "1", ..., "$R-1$", respectively. For a non-root switch S_a (assume $lev(S_a) = L$), find out all paths from each root to S_a, and with respect to each path, assign a locator "$add_0.add_1.....add_L$", where add_0 is the locator of the root, and $add_i (1 \leq i \leq L)$ is the port number of the "level $i-1$" switch that the "level i" switch connects. For each end host H_b, its locators are assigned as "$loc(S_e).P_b(S_e)$", where $loc(S_e)$ is the locator of the edge switch S_e, and $P_b(S_e)$ is the port number of S_e that H_b connects. If there are multiple virtual machines on the same physical machine H_b, then the VMs are assigned locators "$loc(H_b).vmid$". An example of the addressing scheme is given in Figure 1.

The maximal complexity of the tree construction and addressing is at the stage of iterating all switches to find the root candidates, which is $S * O(N+S)$, where S and N are the number of switches and end hosts, respectively. Given that this is a one-off process, optimization is less important.

B. Address Resolution and Forwarding

Locators can be used in different kinds of implementations to provide routing and forwarding for the network, such as IP tunneling or MAC address rewriting ([2], [14]). In the remaining sections, we use MAC address rewriting as the example for our design and experiments.

After assigning locators for all the switches and hosts, the controller records the IP-locators mapping for all the end hosts. And the edge switches record the IP-MAC-locators mapping for the end hosts connected to this switch. When a source host H_r sends out an ARP request for a destination host H_d, the edge switch intercepts the request and forwards it to the controller. Based on the current network situation, the controller selects an appropriate path for this communication. Since multi-path information is indicated by the multiple locators, the chosen path corresponds to one of H_d's locators. The edge switch then returns the corresponding locator to H_r as the ARP reply. In addition, the forwarding tables in the switches are based on destination locators rather than MAC addresses. Therefore, all packet forwarding proceeds based on the locators. Finally, the egress switch performs locators-to-MAC addresses rewriting for the packets and sends it to the destination host H_d. For example, H_5 wants to communicate with H_1. The controller

selects a path "$H_5, S_9, S_5, S_1, S_3, S_7, H_1$" for them, and returns the corresponding locator "0.0.0.0" as ARP reply to H_5. Figure 1 also shows some forwarding table examples. Based on the entries in their forwarding tables, the switches in the path will forward the packet by using "0.0.0.0" as the destination address.

Our design employs a logically centralized controller. The overhead introduced to the network and to the controller by the centralized mechanism mainly comes from the ARP requests/responses between the controller and individual edge switches. Assuming an ARP request or response takes 42 bytes, and each host generates R requests per second, then the traffic overhead is $42 * 8 * R * 2 * N$ bps, where N is the number of end hosts. For example, in an extreme case where each host generates 10 ARP requests per second in a 10,000 host network, the traffic overhead would be 67.2Mbps. In addition, the controller will be processing $R * N$ ARP requests per second. In a large-scale network, the controller CPU could be the bottleneck. But it is possible to move the controller to a small-scale cluster if necessary when high frequency of ARP requests is anticipated ([2]).

IV. Routing and Forwarding

A. Forwarding Model

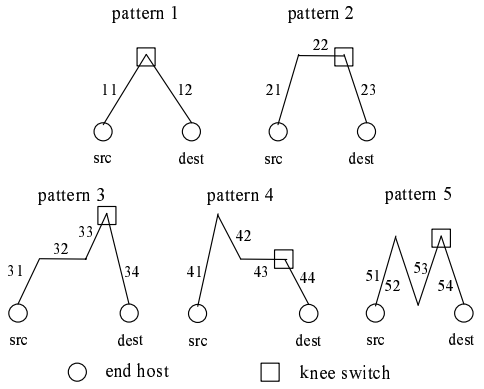


Figure 2. Knee switch examples

In this section, we describe the routing algorithm of how to calculate the forwarding tables. As illustrated in Figure 2, a path from source to destination host may contain several *path segments* of different directions: *upward* (e.g., 11), *downward* (e.g., 12) and *horizontal* (e.g., 22). We name the first switch of the last downward path segment as *knee switch*. In addition, a node x is defined as a *peak switch* of a path if for any other node y in this path, $lev(x) \leq lev(y)$.

An important observation behind our design is that data center networks are often physically inter-connected as a hierarchical structure. In such a network topology, traffic patterns are mostly like this: starting from the source host at the bottom, going up to the access level (ToR or edge switches), and then to the core level (aggregation or core switches), and finally going down to the destination host.

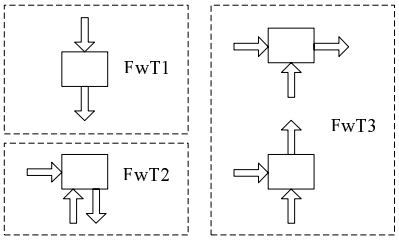


Figure 3. forwarding model

Based on this observation, we apply two constrains for the traffic patterns: *Constraint 1*: Traffic changes direction from *upward* to *downward* no more than once; *Constraint 2*: The *knee* switch must be one of the *peak* switches. Constraint 1 forbids traffic pattern 5 in Figure 2, while Constraint 2 further prevents traffic pattern 4. Finally, only traffic patterns 1, 2, and 3 are allowed. In other words, the path contains one or more *upward* and *horizontal* path segments, to the *peak* switch, and finally a *downward* path segment to the destination. We refer to this as (upward-horizontal)*/(downward)* forwarding, an extension of the (up)*/(down)* forwarding introduced in [15]. One can find that these two constraints work for most of the switch-centric data center network topologies.

According to the two constraints, traffic through a switch S_a can be divided into three forwarding types, *FwT1-FwT3*, as shown in Figure 3.

- *FwT1*: S_a is on the *downward* path segment, subsequent to the *knee* switch. Traffic comes from one *UwP*, and goes out from one *DwP*.
- *FwT2*: S_a is the *knee* switch. Traffic comes from one *DwP* or *PrP*, and goes out from one *DwP*.
- *FwT3*: S_a is on the *upward* or *horizontal* path segment, antecedent to the *knee* switch. Traffic comes from one *DwP* or *PrP*, and goes out from one *PrP* or *UwP*.

B. Multiple Paths

In this section, we show the relationship of multiple paths and multiple locators. The multi-paths between source host H_r and destination host H_b are from three conditions. First, there may be multiple different knee switches. For example, in Figure 1, among the paths from H_1 to H_5, some go through knee switch S_1, and others through S_2. Second, after the knee switch S^K is determined, there may be multiple upward (and/or horizontal) paths between H_r and S^K. For example, if S_1 is S^K, then from H_1 to S_1, one path through S_3 and the other through S_4. Third, after S^K is determined, there may be multiple downward paths between S^K and H_b. For example, from H_5 to H_1, if S_1 is S^K, then there are two downward paths between S_1 and H_1.

loc(H_5)	path: H_1 to H_5
0.2.0.0	S7->S3->S1->S5->S9
	S7->S4->S1->S5->S9
1.2.0.0	S7->S3->S2->S5->S9
	S7->S4->S2->S5->S9

loc(H_1)	path: H_5 to H_1
0.0.0.0	S9->S5->S1->S3->S7
1.0.0.0	S9->S5->S2->S3->S7
0.1.0.0	S9->S5->S1->S4->S7
1.1.0.0	S9->S5->S2->S4->S7

loc(H_3)	path: H_1 to H_3
0.0.1.0 or 1.0.1.0	S7->S3->S8
0.1.1.0 or 1.1.1.0	S7->S4->S8

Figure 4. Multiple locators indicate multiple paths.

According to the addressing mechanism, $loc(S^K)$ is a prefix of $loc(H_b)$, i.e., $loc(H_b) = loc(S^K).loc'(H_b)$. For the first kind of multi-paths, choosing different S_j^K will bring different $loc(S_j^K)$, so the corresponding $loc(H_b)$ is different. For example, as shown in Figure 4, H_1 communicates with H_5. $loc(H_5)$ is chosen as *0.2.0.0* or *1.2.0.0* to indicate different knee switches S_1 and S_2 respectively. Given one knee switch S_j^K, if it has multiple locators, then the corresponding $loc(H_b)$ are different. However, the different $loc(H_b)$ represents the same path. In the example of H_1 to H_3, S_3 has two locators, and correspondingly, H_3 gets two locators *0.0.1.0* and *1.0.1.0*. However, these two locators represent the same path.

For the second kind of multi-paths, multiple upward (and/or horizontal) paths share the same $loc(H_b)$, since locators encode the downward-path information from the root to the node. In the example of H_1 to H_5, when S_1 is the knee switch, the two paths from H_1 to S_1 correspond to the same locator. Path selection for this kind of multi-paths is done in switch S_7. As shown in the forwarding table of S_7 in Figure 1, packets can be forwarded from both port 2 and 3 when the destination locator is "0.*".

For the third kind of multi-paths, choosing different downward path from S^K to H_b will bring different $loc'(H_b)$, so the corresponding $loc(H_b)$ is different. In the example of H_5 to H_1, if S_1 is S^K, then the two downward paths between S_1 and H_1 correspond to two locators *0.0.0.0* and *0.1.0.0*. In summary, multiple paths are related to multiple locators. However, the relationship is not one-to-one mapping. Multiple locators may represent one path, and multiple paths may share one locator.

C. Routing Algorithm

Next, we introduce the routing algorithm to calculate the forwarding table for a switch S_a given the destination host H_b. To forward a packet, a switch first needs to decide the direction of the output port. If H_b is a descendant of S_a, and from a downward port $DP_j(S_a)$ there is a downward path to H_b, then S_a should forward the packet downward from $DP_j(S_a)$. According to the addressing mechanism, $loc(S_a).DP_j(S_a)$ is a prefix of $loc(H_b)$. Therefore, we set the forwarding rule $loc(S_a).DP_j(S_a).* \Rightarrow DP_j(S_a)$ for each $DP_j(S_a)$. This rule handles $FwT1$ and $FwT2$.

$FwT3$ means that S_a is on the *upward* or *horizontal* path segment, before the *knee* switch. To handle this, the algorithm performs breadth-first traversal to *upward* and *horizontal* direction from S_a to the root of the multi-rooted tree. In other words, the breadth-first traversal is only from S_a's *UwP* and *PrP*, not from its *DwP*. After this step, all the switches in S_a's *upward* and *horizontal* direction and the corresponding port of S_a to reach that switch are added into a set $sset$. Finally, for each $(S_m, P_m(S_a))$ in $sset$, we set the forwarding rule $loc(S_m).* \Rightarrow P_m(S_a)$. The forwarding rules for S_a are summarized below:

$$loc(H_b) = loc(S_a).DP_j(S_a).* \Rightarrow DP_j(S_a) \quad (1a)$$

$$loc(H_b) = loc(S_m).* \Rightarrow P_m(S_a) \quad (1b)$$

A forwarding table example has been shown in Figure 1. The time complexity of the routing algorithm is $O(S*O(\frac{P}{2})^{\frac{d}{2}})$, where S is the number of switches, P is the number of ports per node, and d is the average path length. The algorithm only does partial bread-first traversal from a switch's *UwP* and *PrP* to the root, so both the port number and path length become half of the original value.

The hierarchical addressing mechanism leads to aggregation of forwarding entries. Equation (1a) and (1b) indicate aggregation for the descendants of switch S_a and S_m respectively. After all forwarding entries are generated, some entries from (1b) can be further merged. Assume we have two entries $loc(S_{m1}).* \Rightarrow P_m(S_a)$ and $loc(S_{m2}).* \Rightarrow P_m(S_a)$, and $loc(S_{m1})$ is a prefix of $loc(S_{m2})$, then they can be merged as $loc(S_{m1}).* \Rightarrow P_m(S_a)$.

Since in address resolution phase any one of the multiple locators may be chosen, so all the locators for a node need to be recorded in the forwarding entries. As shown in Equation(1a) and (1b), if switch S_a (or S_m) has multiple locators, then there will be multiple forwarding entries. For example, switch S_7 has 4 locators, so Equation (1a) generates 4 forwarding entries for port 0 and 1. Assume in the bread-first traversal from a switch S_a's *UwP* and *PrP* to the root, there are S^U switches, then Equation (1b) generates $S^U * M$ entries, and each DwP generates M entries. Therefore, the forwarding state of a switch S_a is $O(S^U * M + M * P/2)$, where P is the number of ports per node, and M is the number of multiple paths.

D. Reducing Forwarding States

Tracking all locators will cause switches have a very large forwarding tables. To reduce the redundant forwarding state, we utilize input port in forwarding, in addition to the destination address. If the input port is one of its UwP, then it is *FwT1*; otherwise, it is *FwT2* or *FwT3*. For *FwT1*, the only thing S_a need to do is to decide from which *DwP* to send out the packets. According to the addressing mechanism, $loc(H_b) = loc(S_a).DP_j(S_a).*$, where $DP_j(S_a)$ is the port number of S_a that connects to H_b or H_b's ancestor. We use L to denote $lev(S_a)$, then S_a can decide the *DwP* by looking at $add_{L+1}(H_b)$, the

$(L+1)^{th}$ field of $loc(H_b)$. Therefore, the forwarding rule can be simplified to $add_{L+1}(H_b) = DP_j(S_a) \Rightarrow DP_j(S_a)$.

For *FwT2* or *FwT3*, an ideal situation is only one locator of a switch is recorded, rather than tracking all locators of S_a and S_m. This needs some cooperation of the controller in the locator selection process. The controller first chooses one knee switch S_j^K. Assuming S_j^K has L_j locators: $loc_l(S_j^K)$, then some of H_b's locators are formatted as $loc(H_b) = loc_l(S_j^K).loc'(H_b)$. Then the controller selects one downward path, which corresponds to one $loc'_k(H_b)$. Finally, the controller decides "$loc_0(S_j^K).loc'_k(H_b)$" to be the locator of host H_b, where $loc_0(S_j^K)$ represents the first locator of $loc(S_j^K)$. As mentioned, given a chosen knee switch, its multiple locators will bring different $loc(H_b)$. However, they actually represent the same path. Therefore, if we use the first locator in the controller and the forwarding table, there will be no loss of multiple-path capacity.

In summary, forwarding rules in Equations (1a) and (1b) are converted to Equations (2a)-(2c) shown below. Here, $loc_0(S_a)$ and $loc_0(S_m)$ represents the first locator of $loc(S_a)$ and $loc(S_m)$. Equation (2a)-(2c) correspond to forwarding type *FwT1-FwT3* respectively. The reduced forwarding state is $O(S^U + P)$, much smaller than the original value $O(S^U * M + M * P/2)$.

$$InPort = UwP, add_{L+1}(H_b) = DP_j(S_a) \Rightarrow DP_j(S_a) \quad (2a)$$

$$loc_0(S_a).DP_j(S_a).* \Rightarrow DP_j(S_a) \quad (2b)$$

$$loc_0(S_m).* \Rightarrow P_m(S_a) \quad (2c)$$

V. Other Components

The multi-locator mechanism brings built-in support for multi-path routing, load balance and fault tolerance. Here we describe other components of our system, including scheduling, fault tolerance and mobility support.

One of the approaches to perform flow scheduling is to insert forwarding entries into switches on the fly. While our system carries out scheduling by choosing different locators in the address resolution phase, thus no forwarding table modifications are required to the switches along the path. For example, in the topology shown in Figure 1, both H_5 and H_7 communicate with H_1. If we would like the first pair takes the path "$H_5, S_9, S_5, S_1, S_3, S_7, H_1$", and the second pair takes the path "$H_7, S_{10}, S_5, S_2, S_4, S_7, H_1$", then the controller can assign locator "0.0.0.0" for H_1 and returns to H_5, and assign locator "1.1.0.0" for H_1 and returns to H_7.

Various scheduling mechanisms can be used in selecting appropriate paths. A straightforward approach could be as follows: switches periodically report traffic status of each link to the controller, then the controller has a global view of the link utilization. In path selection, the controller finds the bottleneck links for each path between the host pair, and selects the path with the most light-loaded bottleneck links.

However, during ARP resolution, a host expects only one MAC address for a particular IP address, and stores it in its ARP cache. To dynamically change the path for an on-going communication, the controller will send an address translation rule to the ingress switch, which includes "*srcIP*, *destIP*, *destLoc*". Then the ingress switch will rewrite the destination MAC address from the original destination locator to this new *destLoc* for the packets between *srcIP* and *destIP*, thus the communication path will be changed. To provide flow level scheduling, the address translation rule also need to consist port number information.

Our system provides natural fault tolerant routing based on the selection of multiple locators. After detecting a link failure, the controller disables the paths that pass through the failed link by returning other active locator when receiving an ARP request. To provide fault tolerance for the on-going communications, the controller chooses another available path and sends the corresponding address translation rule to the ingress switch. Mobility support is similar to fault tolerance. For example, a virtual machine migrates from a physical machine to another. When the edge swtich detects the new VM, it reports the change to the controller. The controller then assigns new locators to the VM, and returns new ones for the new ARP request about the VM. For the on-going communications, the controller needs to send address translation rules to all the edge switches that connect a source host communicating with the VM.

VI. Simulations

In this section, we first generate a variety of network topologies to evaluate the forwarding states, and then evaluate several scheduling algorithms using different traffic matrixes. The simulated topologies include: 1) Fat-tree(p)([6]): p is the number of ports per switch. 2) VL2(p)([1]): p is the number of ports per intermediate switch, and the number of servers per rack is 20. 3) CiscoDC(m, a)([5]): a three-layer tree with two core switches, m is the number of aggregation modules, and a is the number of access switch pairs associated with each aggregation module.

A. Forwarding States

Forwarding states of different network topologies are shown in Table II. Columns "#paths", "fws" and "fws (reduced)" are the number of multi-paths between host pairs, the original forwarding states and the reduced forwarding states, respectively. One can find that the basic algorithm generates a large number of forwarding states when the number of multi-paths is large, while the reduced one generates a small number of forwarding states in most cases.

In fat-tree topology, a edge switch need to store one forwarding entry for each of the $p^2/4$ core switches, and the forwarding entries cannot be aggregated. Therefore, its forwarding state is relatively large for fat-tree(48). While in VL2 topology, the number of core switches is $p/2$, so the reduced forwarding states for all switches is very small. In CiscoDC topology, according to the connection pattern, the

Topology	#hosts	#switches	#paths	fws			fws (reduced)		
				core	agg	edge	core	agg	edge
Fat-tree(8)	128	80	16	8	20	80	8	12	24
Fat-tree(24)	3456	720	144	24	156	1872	24	36	168
Fat-tree(48)	27648	2880	576	48	600	14400	48	72	624
VL2(4)	80	10	8	4	6	86	4	6	44
VL2(48)	11520	648	96	48	600	1032	48	72	66
VL2(96)	46080	2448	192	96	2352	2064	96	144	90
CiscoDC(2,2)	192	14	8	5	12	106	5	11	53
CiscoDC(8,8)	3072	146	8	17	36	106	17	35	53
CiscoDC(16,16)	12288	546	8	33	68	106	33	67	53

Table II
FORWARDING STATES FOR DIFFERENT TOPOLOGIES

numbers of locators and multi-paths do not increase with m and a. Therefore, both the basic and reduced the algorithm generate very small forwarding tables in large scale network.

B. Scheduling

Different scheduling methods are evaluated under a group of communication patterns, similar to [16]: 1) *Stride(i)*: A host with index x sends to the host with index $(x + i)$ *mod (num_hosts)*. 2) *All-to-all*: Every host sends to every other hosts in the network. 3) *Random*: A host sends to any other host in the network with uniform probability. In each pattern, every host sends out one 1GB file in one TCP flow, and the time interval of the start time of each flow is 1ms. Three scheduling mechanisms are evaluated: 1) *Fixed locator*: fixedly return the first locator of the destination host for every source host, which simulates the spanning tree protocol; 2) *Random locator*: randomly select one locator from the multiple locators of the destination host; 3) *Traffic-aware locator*: select a locator based on the link status of the multiple paths. We simply use the number of flows in each link to indicate the link status. Since in the traffic matrixes, all hosts send out one flow of the same size, this simple method achieves near optimal aggregate throughput.

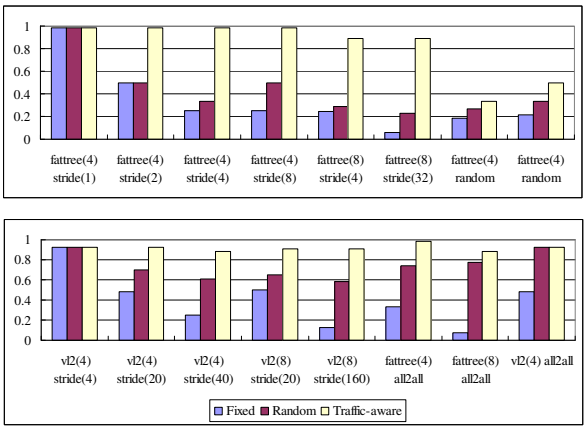


Figure 5. Aggregate throughput for three communication patterns on various network topologies.

Some of the evaluation results are shown in Figure 5. To put the results for different topologies in one figure, we normalize the aggregate throughput by dividing by the non-blocking throughput of the network. From the results, one can find that traffic-aware locator selection significantly outperforms the others. For small step stride patterns, most of the communications happen under the same edge switch, so all the three scheduling method produce very high throughput, e.g., fat-tree(4)/stride(1) and vl2(4)/stride(4). Random locator selection performs better when the number of flows or the number of multi-paths is large, e.g., in fat-tree(8)/all-to-all and vl2(4)/all-to-all. The aggregate throughput in fat-tree(4)/random is low, since several source hosts may randomly select the same destination host.

VII. PROTOTYPE EVALUATION

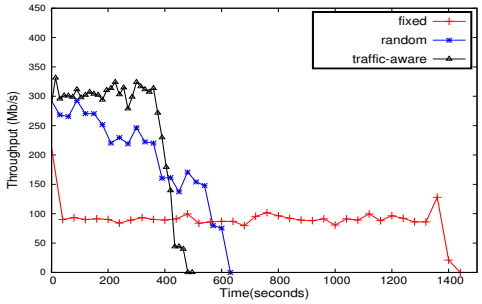


Figure 6. Comparison of different scheduling algorithms

We built a testbed of fat-tree(4) topology, which consists of 10 DELL Optiplex 990 desktops and 16 Intel PRO/1000 PT Quad Port Ethernet NICs. We deploy the 36 elements (20 switches and 16 hosts) of fat-tree(4) topology in 10 physical machines. The four switches in one pod are hosted in one machine by running four virtual machines; and each pod's four end hosts are hosted in one machine. The remaining two machines are running two VMs in each to host four core switches. In addition, the machines are plugged into one or two quad port NICs to provide appropriate number of interfaces for the switches and hosts. For the four switches

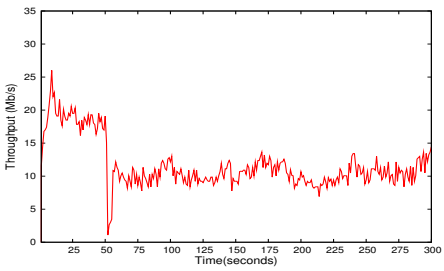


Figure 7. Response to link failure

in the same pod, we use the virtual links provided by the hypervisor to connect them. The controller is run in another PC, connected to the switches by a 24-port Gigabit TP-Link switch. We use *ebtables* to implement the forwarding table, by mapping each forwarding entry to an *ebtables* configured filter rule. The edge switches also add "locator ↔ MAC" NAT rule to do MAC/locator addresses rewriting.

Figure 6 shows the aggregate throughput of the three locator selection schemes for stride(4) traffic matrix. The fixed locator has very limited total throughput (90Mbps) since all the flows from one pod choose the same uplink to the core switch. The random locator has an obviously improved total throughput (196.8Mbps), and the traffic-aware locator has the best total throughput (256Mbps).The reason of the low throughput in the testbed is that the virtual links between the VMs in one physical machine are set to be 100Mbps, and virtual machines bring performance overhead.

We further evaluate the fault tolerance of our system, by dropping one link between an aggregate and a core switch at the time point of 50 second after all the flows have started. Figure 7 shows the throughput for one flow affected by the dropped link. The throughput dropped to zero at that time point and react in slow-start mode in the new path. Its throughput decreases nearly 50% because the rescheduled flow competes with other original flows in the chosen link.

VIII. Conclusion

In this paper, we propose a generic addressing and routing protocol for hierarchical data center networks. We first form the topology as a multi-rooted tree, and then assign each switch or host one or more locators. Each locator encodes a downward path from the roots to this node. When there are a large number of multi-paths in the network, some switches may have a large forwarding table, since switches need to store multiple forwarding entries for multiple locators. We further introduce a new forwarding model, which generates very small forwarding tables, and at the same time keeps the multi-path capability. We use simulation to evaluate our system on a variety of network topologies with different sizes, and also build a testbed of fat-tree topology consisting 16 hosts. The evaluation compares the forwarding state for various topologies, and the aggregate network throughput using different scheduling methods. In addition, the testbed experiments show that our system has good fault tolerance.

References

[1] 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," *SIGCOMM 2009*.

[2] R. 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," *SIGCOMM 2009*.

[3] C. Guo, H. Wu, K. Tan, L. Shi, Y. Zhang, and S. Lu, "Dcell: A scalable and fault-tolerant network structure for data centers," *SIGCOMM 2008*.

[4] C. Guo, G. Lu, D. Li, H. Wu, X. Zhang, and etc., "Bcube: A high performance, server-centric network architecture for modular data centers," *SIGCOMM 2009*.

[5] "Cisco data center infrastructure 2.5 design guide," *http://www.cisco.com/application/pdf/en/us/guest/netsol/ns107/c649/ccmigration_09186a008073377d.pdf*.

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

[7] T. L. Rodeheffer, C. A. Thekkath, and D. C. Anderson, "Smartbridge: A scalable bridge architecture," *SIGCOMM 2000*.

[8] S. Sharma, K. Gopalan, S. Nanda, and T. cker Chiueh, "Viking: A multi-spanning-tree ethernet architecture for metropolitan area and cluster networks," *INFOCOM 2004*.

[9] "Ietf trill," *http://datatracker.ietf.org/wg/trill/charter/*.

[10] J. Mudigonda, P. Yalagandula, M. Al-Fares, and J. C. Mogul, "Spain: Cots data-center ethernet for multipathing over arbitrary topologies," *NSDI 2010*.

[11] C. Kim, M. Caesar, and J. Rexford, "Floodless in seattle: A scalable ethernet architecture for large enterprises," *SIGCOMM 2008*.

[12] K. Chen, C. Guo, H. Wu, J. Yuan, Z. Feng, Y. Chen, S. Lu, and W. Wu, "Generic and automatic address configuration for data center networks." *SIGCOMM 2010*.

[13] M. Walraed-Sullivan, R. N. Mysore, M. Tewari, Y. Zhang, K. Marzullo, and A. Vahdat, "Alias: Scalable, decentralized label assignment for data centers," *SOCC 2011*.

[14] M. Scott, D. Wagner-Hall, A. Moore, and J. Crowcroft, "Addressing the scalability of ethernet with moose," *DC CAVES Workshop 2009*.

[15] M. Schroeder, A. Birrell, M. Burrows, H. Murray, R. Needham, and etc., "Autonet: A high-speed, self-configuring local area network using point-to-point links," *IEEE Journal on Selected Areas in Communications*.

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