VCN: Versatile Clos-type Networks for Traffic Locality in Data Centers

Jun Duan and Yuanyuan Yang
Department of Electrical and Computer Engineering, Stony Brook University, Stony Brook, NY 11794, USA

***Abstract*—Traffic locality is ubiquitously exploited in today's data centers. It allows network resources to be used more efficiently because traffic in data centers is adapted to the underlying network infrastructures. In this paper, we approach this problem from another direction, which is to adapt network infrastructure to the traffic. Towards this direction, we adopt two approaches to boost the efficiency of network resource utilization. First, we improve the topology of Clos-type data center networks by introducing horizontal connections, which facilitates the routing of local traffic. Second, based on the improved topology, we make the network infrastructure to be versatile, in the sense that it can be fitted into the characteristics of the traffic. We name this design as Versatile Clos-type Networks (VCN). We describe the topological architecture of the VCN, design its addressing and routing schemes, and demonstrate its various properties. We also evaluate its performance and compare it with fat-tree, a representative Clos-type data center network architecture. The evaluation results show that the VCN can deliver the same throughput as fat-tree, but use significantly reduced network resources.**

Keywords*—Data center networks, Clos-type networks, traffic locality, network architecture.

I. Introduction

Clos-type networks are a family of interconnection networks which are named after a well known study by [1]. Although the result of [1] is intended to improve the performance of telephone switching systems, its variations have found wide applications in multicast communications [12]–[14], [20]–[22], supercomputing [2] and cloud computing [5]. Especially, they are ubiquitously deployed in today's data centers as the fabric which provides interconnections to enormous number of servers [3], [4], [6], [7].

A. Overview of Clos-type Data Center Networks

Clos-type networks are the de facto standards for data centers. Initially, Clos-type data center networks are designed in a *hierarchical* manner. That is, the network uses a progressively fewer number of, but more specialized and expensive equipments moving up from the bottom tier to the top tier in the structure of the network. This design paradigm was suggested about a decade ago by major network device vendors such as Cisco [8]. As a concrete example, some of Facebook's current data centers are still using a "4-post" Clos design, which is detailed in [9]. This network is composed of three tiers of switches, named RSW (top-of-rack switch), CSW (cluster switches) and FC (FatCat), from bottom to top. In each cluster, there are only four CSWs which are in charge of all intra-cluster and inter-cluster traffic. Similarly, in each data center building, there are only four huge and expensive FCs as well.

The other design paradigm of the Clos-type data center networks, in comparison with the hierarchical paradigm, adopts uniform network devices in all the tiers of the structure. Instead of a few large, high-end and expensive switches in the upper tiers, this paradigm uses a relatively larger number of commercial off-the-shelf and inexpensive switches throughout the entire network. The gap (in terms of port density, line speed, price etc.) between the switches in the upper tiers and the lower tiers is eliminated, which is fundamentally different from the hierarchical designs. A representative of the these designs is fat-tree [3]. In fat-tree, all the switches in the network are identical. These identical switches are arranged in three tiers, namely, the edge tier, the aggregation tier and the core tier, from bottom to top, as shown in Fig. 1.

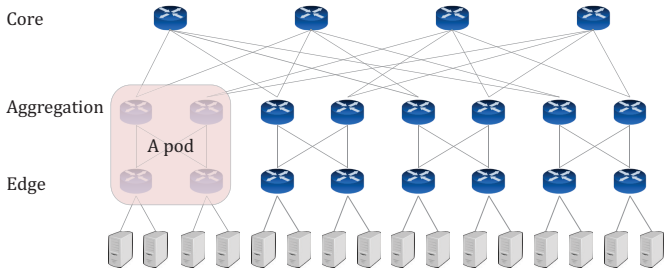


Fig. 1. An example of Clos-type network: a 4-ary fat-tree. The network consists of three tiers of uniform, commercial off-the-shelf switches: the edge tier, the aggregation tier, and the core tier. In the network, no link connects two switches within the same tier.

From both academic literatures and industrial deployments, we can find that data center network infrastructures are evolving overtime. One of the most important trends is that, cloud providers are gradually transferring from the hierarchical Clos-type networks to fat-trees. Takes Cisco's data center as an example again: About ten years ago, Cisco's design guide suggested the hierarchical designs for its customers [8]. But nowadays, we notice that in its own newly deployed data centers, off-the-shelf components are used "as much as possible" [25]. Also, for Facebook's data centers, it is stated in [9] that its data centers' architectures are constantly evolving, and fat-tree [3] is especially emphasized as a possible alternative for the existing 4-post deployment. Actually, Facebook recently disclosed its next generation network architecture, which largely relies on commercial off-the-shelf switches, which are "much simpler than the large switches before" [24].

B. Traffic Locality in Data Centers

Traffic locality is a key factor that has considerable impact on the performance of distributed processing in data centers [10], [11], [15], [16]. For example, in Apache Hadoop [10], it is proved that the locality-oblivious assignments of reduce tasks can result in performance degradation [11]. The authors of [11] then propose an improved reduce task scheduler called

978-1-5386-2542-2/18/$31.00

LARTS, which takes the size and location of the input data into consideration before the assignments of reduce tasks.

Aside from the performance considerations, traffic locality also has decisive impact on the cost in data centers. For this reason, virtual machines (VMs) with large mutual bandwidth usages are usually placed in close proximity to improve the scalability of the networks [17]. Furthermore, it is suggested that, traffic locality can affect not only the cost caused by network traffic (N-cost), but also the cost caused by the utilization of physical machines (PM-cost) [18]. Therefore, in addition to efficient VM placement, researchers consider more techniques to deal with traffic locality. In [19], the authors propose VMPlanner, which optimizes both virtual machine placement and traffic flow routing so as to turn off as many unneeded network elements as possible for power saving. In [23], virtual machine migration is used in addition to virtual machine placement, to minimize the data transfer time consumption and avoid congestion.

Furthermore, the proportion of localized traffic is becoming increasingly significant in today's data centers. For example, the machine-to-machine traffic in Facebook's data centers are growing at much higher rate than that of the machine-to-user traffic [24], which suggests that the servers inside a data center are collaborating more and more closely.

C. Problem Statement

We have seen that traffic locality leads to efficient usage of the network resources. Driven by cost and performance considerations, the traffic is *adapted* to network infrastructures so that the usage of the network resources are reduced. Besides that, we have also seen that the network infrastructures themselves are evolving from hierarchical designs to fat-tree, as reviewed in Section I-A. There could be a number of reasons to explain the evolution from hierarchical design to the fat-tree. For example, the fat-tree has richer path redundancy, higher degree of fault tolerance, lower management complexity, etc. Particularly, in the hierarchical design, the bandwidth provisioning at upper tiers are usually insufficient. It is exactly motivated by this lack of bandwidth provisioning [3], that fat-tree is proposed. With fat-tree, for the first time it is possible to deliver aggregate bandwidth at 1:1 oversubscription between all adjacent tiers of switches, wherever at the edge or core. In other words, the evolution of the network infrastructure follows the increasing bandwidth demands of the traffic in data centers. From this point of view, we have an observation that the evolution of the network infrastructure also *adapts* to the characteristics of the traffic, but in much slower pace.

Increasing bandwidth demands, of course, only partially characterize the traffic in data centers. Another important characteristic of the traffic that we can't ignore is the aforementioned traffic locality. Therefore, a natural question arises here: Aside from the bandwidth demands, *how will the network infrastructures adapt to the ubiquitous traffic locality?* To be specific, given that the traffic is localized, can we identify an efficient way to deploy the network infrastructures? We can visualize this problem in Fig. 2. Considering traffic locality, the bandwidth demand in the network is trapezium-shaped throughout the multiple tiers. The hierarchical design is clearly not the optimal solution for the traffic because it cannot provide sufficient bandwidth at upper tiers. The

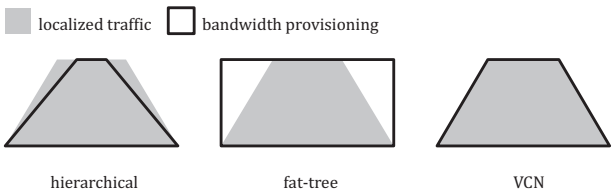


Fig. 2. Adapting network infrastructures to the localized traffic in data centers. At the upper tiers of switches, hierarchical designs are under-provisioning and fat-tree is over-provisioning.

fat-tree, however, is actually over-provisioning at upper tiers because of its undifferentiated 1:1 oversubscription. Hence, both of them are oblivious of traffic locality. What we need is a fitted provisioning for localized traffic in data centers, which motivates us to the design of a novel network architecture: *Versatile Clos-type Networks (VCN)*.

D. Challenges and Contributions

To design such a fitted network, there are at least two challenges. The first challenge is: If the network architecture is fitted to the traffic, then there will be less redundant network resources comparing to the fat-tree, especially at upper tiers. In this case, how can we avoid traffic congestions, e.g., caused by traffic fluctuations? The second challenge is: How to quantitatively describe the degree of traffic locality so that we can precisely fit the network infrastructures to the localized traffic? In this paper, we manage to resolve these two challenges by proposing Versatile Clos-type Networks (VCN), for data centers. The contributions of the paper include:

- We introduce horizontal links into Clos-type networks to facilitate the routing of local traffic. In this way, redundant network resources are provided to non-local traffic. This redundancy acts as a "cushion" between the traffic and the fitted network infrastructure, therefore, congestion is avoided and the first challenge is resolved.
- We design a fast yet accurate way to define the degree of traffic locality and fit the network topology to the localized traffic to resolve the second challenge.

The rest of the paper is organized as follows. In Section II, we review some related works. In Section III, we present the topology of the VCN, including how we deploy horizontal connections. After that, in Section IV, we introduce the addressing and routing schemes for the VCN. Then, we describe how to fit the topology of VCN into the traffic in Section V. We also evaluate the performance of the VCN and make comparisons in Section VI. Finally, Section VII concludes this paper.

II. Related Work

Clos-type networks [3]–[7], [12] share some topological similarities, such as a multi-rooted, multi-tier structure, which delivers rich connectivity and large bisection bandwidth. Generally, a Clos-type network is composed of multiple tiers of switching elements (as shown in Fig. 1), along with links that interconnect the switching elements. An important feature of a Clos-type networks is that, a link always connects two switching elements which belong to two adjacent tiers. That is,

TABLE I. LIST OF FREQUENTLY USED NOTATIONS

notation	description
Topology of VCN	
h_i	horizontal index of edge switches
h_j	horizontal index of aggregation switches
i	edge offset
j	aggregation offset
k	number of ports on each switch
Addressing	
p	index of the pods in a VCN, $p = 0, 1, \ldots, k-1$
a	index of the aggregation switches in a pod
e	index of the edge switches in a pod

there is no such links which connect two switching elements within the same tier. Figuratively speaking, the links in the network all carry up and down traffic between tiers vertically. There are no horizontal links.

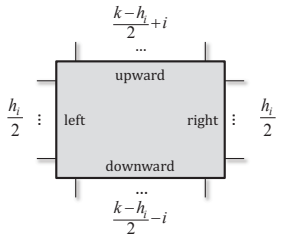
The only exception we know comes from the aforementioned Facebook's 4-post design. In the 4-post design, the four CSWs in each cluster are connected in a "protection ring," and the four FCs in each data center are also connected in a protection ring [9]. However, as the name suggests, these connections are made primarily for the fault tolerance considerations. Since there is a very limited number of high-end CSWs/FCs in this hierarchical design, the network is vulnerable to device failures. According to [9], a CSW failure reduces intra-cluster capacity to 75%, and an FC failure reduces inter-cluster capacity to 75%. This vulnerability is a major disadvantage of the 4-post design. Considering the disadvantage, Facebook is now developing its new generation of data center network design [24], which is also a Clos-type network. In this newer generation, we do not spot those horizontal protective connections.

On the other hand, comparing to a hierarchical design such as the 4-post structure, fat-tree possesses several attractive features [3]. First, by leveraging extra tiers of switches, fat-tree provides significantly higher scalability. Second, by using the commodity network devices, the maintenance complexity is greatly reduced. Last, with richer path redundancy, fat-tree is not sensitive to device failures, thus offers higher degree of fault tolerance. Since in a fat-tree, all the switches have the same number of ports at the same link speed, a fat-tree can be completely defined by k, the port count of one switch, and be denoted by a k-ary fat-tree. For example, Fig. 1 shows the complete structure of a 4-ary fat-tree. It delivers a uniform 1:1 oversubscription between each pair of adjacent tiers.

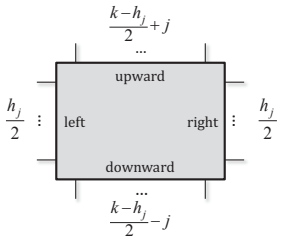
III. TOPOLOGICAL MODEL

Similar to the fat-tree, the building blocks of the VCN are uniform, which are commercial off-the-shelf k-port switches. The switches are arranged in three tiers. For consistency considerations, we also name them as the edge tier, the aggregation tier, and the core tier. Aside from the switches, the fabric of the VCN is also composed of two types of links: the vertical links which connect switches from adjacent tiers, and the horizontal links which connect switches within tiers.

In our topological model, horizontal links may exist within the edge tier or the aggregation tier. For each edge switch, we use h_i out of the k ports for the horizontal links. We name the parameter h_i as horizontal index of edge switches. Similarly,



(a) edge switch



(b) aggregation switch

Fig. 3. Port allocation of edge switches and aggregation switches. Every edge or aggregation switch has k ports, among which h_i or h_j ports are used for horizontal links.

we define h_j, the horizontal index of aggregation switches. Frequently used notations in this paper are listed in Table I.

Vertical links exist between the servers and the edge tier, the edge tier and the aggregation tier, and the aggregation tier and the core tier. For each edge switch, we use $(k-h_i)/2-i$ ports for downward links, i.e., links to the servers, and $(k-h_i)/2+i$ ports for upward links, i.e., links to the aggregation switches. For each aggregation switch, we use $(k-h_j)/2-j$ ports for downward links, i.e., links to the edge switches, and $(k-h_j)/2+j$ ports for upward links, i.e., links to the core switches. The two parameters i and j depict the difference of port count for the upward links and downward links, thus we name them edge offset and aggregation offset, respectively. The port allocation of the edge switches and the aggregation switches are illustrated in Fig. 3. For each core switch, all its k ports are used for vertical links, i.e., links to the aggregation switches.

In the VCN, edge switches and aggregation switches can be partitioned into strongly connected components, namely, pods. In each pod, every edge switch has exactly one bidirectional link to every aggregation switch. These links fall in the category of aforementioned vertical links. Besides, edge switches also use their left and right ports (as shown in Fig. 3(a)) to setup horizontal links between neighbor edge switches inside the pods.

On top of the pods, a tier of core switches finalizes the topology of the VCN. In the VCN, every pod has exactly one bidirectional link to each of the core switch. In addition to these vertical links, aggregation switches also use their left and right ports (as shown in Fig. 3(b)) to setup horizontal links between neighbor pods. In the design of VCN, there are no horizontal links in the core tier. Core switches do not have left or right ports, and all of their ports are used for vertical links going downwards to the aggregation tier.

The aforementioned five parameters completely define the topology, thus we can use a tuple (h_i, h_j, i, j, k) to specify

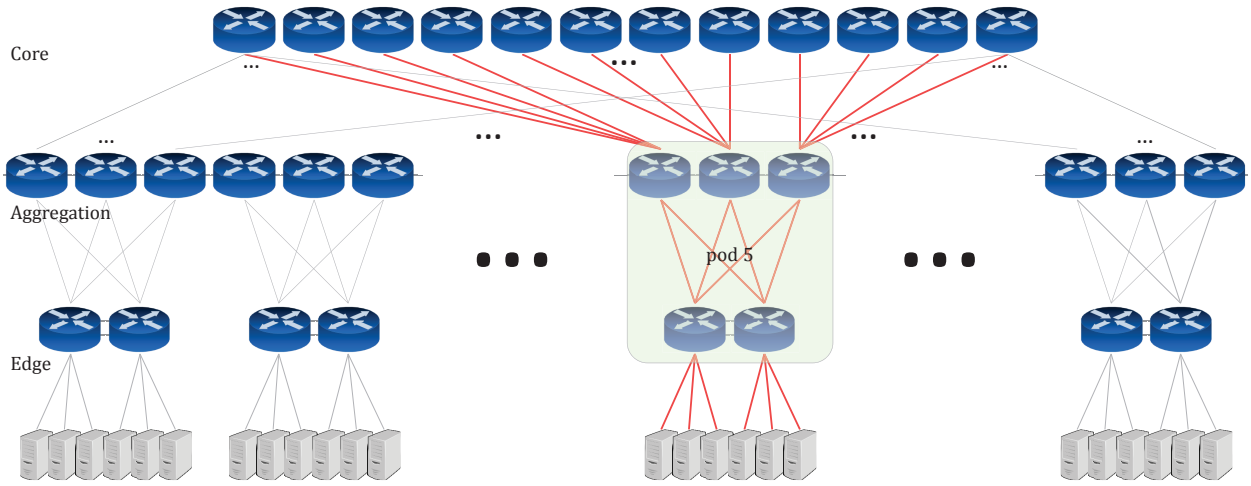


Fig. 4. Topology of an $(h_i = 2, h_j = 2, i = 0, j = 1, k = 8)$ VCN. It is composed of three tiers (edge, aggregation and core) switches and inter-tier, vertical links and intra-tier, horizontal links.

TABLE II. AN EXAMPLE TRAFFIC MATRIX OF 4 COLLABORATING SERVERS A-D

	A	B	C	D
A	-	-	1 Gbps	-
B	-	-	-	1 Gbps
C	1 Gbps	-	-	-
D	-	1 Gbps	-	-

a VCN. If we set horizontal indices and offsets to zeros, we can clearly see that the topology becomes a traditional fat-tree. That is, a $(0, 0, 0, 0, k)$ VCN is equivalent to a k-ary fat-tree. Therefore, the fat-tree can be seen as a special case of the VCN. We compare the topologies of a (h_i, h_j, i, j, k) VCN and a k-ary fat-tree in Table III. We also show an example $(2, 2, 0, 1, 8)$ VCN in Fig. 4.

Next, we use a simple example to demonstrate the benefit of introducing horizontal connections. Consider a pod in a 4-ary fat-tree as shown in Fig. 5(a). It includes two edge switches 1 and 2, two aggregation switches 3 and 4, and connects four servers, from A to D. Assume that the four servers are collaborating closely, and the traffic matrix among them is shown in Table II. That is, servers A and C simultaneously send data to each other, both at data rate of 1 Gbps. The same is true for servers B and D. This matrix is simple, but the corresponding traffic has actually reached the maximal deliverable throughput of this pod which is 4 Gbps, given that the switches and servers are connected by GigE links.

In these settings, one way to route the traffic is shown in Fig. 5(a). The solid lines represent the path for the bidirectional traffic between servers A and C. The dashed lines are for servers B and D. In other words, the aggregation switch 3 is in charge of the traffic between servers A and C, and the aggregation switch 4 between servers B and D. Another way is to switch the roles of switches 3 and 4, but in both ways, the two aggregation switches must be used. Different from Fig. 5(a), if we introduce a horizontal connection between switches 1 and 2, the same traffic can also be routed. This time, however, we can save one aggregation switch, for example, the aggregation switch 4 in Fig. 5(b).

Horizontal links may also exist in the aggregation tier. In

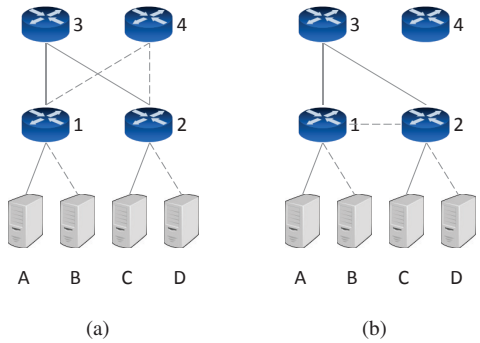


Fig. 5. A simple example demonstrating the benefit of horizontal connections. Solid lines route the traffic between servers A and C. Dashed lines route the traffic between servers B and D. Using a horizontal link between switch 1 and switch 2, the traffic matrix in Table II can be fulfilled without using switch 4.

TABLE III. TOPOLOGICAL COMPARISON BETWEEN THE VCN AND THE FAT-TREE

item	(h_i, h_j, i, j, k) VCN	k-ary fat-tree
# core switches	$\left(\frac{k-h_i}{2} + i\right)\left(\frac{k-h_j}{2} + j\right)$	$\frac{k^2}{4}$
# aggregation switches	$\left(\frac{k-h_i}{2} + i\right) k$	$\frac{k^2}{2}$
# edge switches	$\left(\frac{k-h_j}{2} - j\right) k$	$\frac{k^2}{2}$
# pods	k	k
# servers	$k\left(\frac{k-h_i}{2} - i\right)\left(\frac{k-h_j}{2} - j\right)$	$\frac{k^3}{4}$

Fig. 5, we see that edge switches use horizontal links to reach their counterparts which are in the same pod but at different positions. In this way, in the edge tier, we have horizontal cycles inside each pod. Similarly, in the aggregation tier, the switches use horizontal links to reach their counterparts which are in the same VCN but in different pods. For example, horizontal connections of pod 5 in the aggregation tier in an $(2, 2, 0, 1, 8)$ VCN are shown in Fig. 6. Similarly, we have horizontal cycles inside the VCN at the aggregation tier.

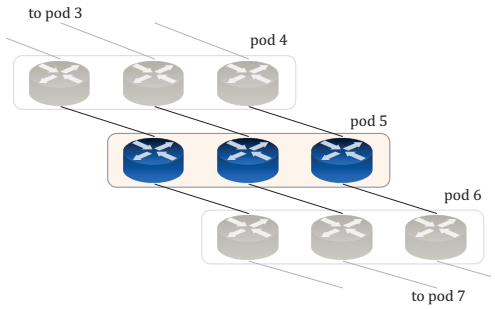


Fig. 6. Horizontal connections of pod 5 in the aggregation tier of a (2,2,0,1,8) VCN. Each aggregation switch has one horizontal link to its counterpart in pod 4 and another horizontal link to its counterpart in pod 6.

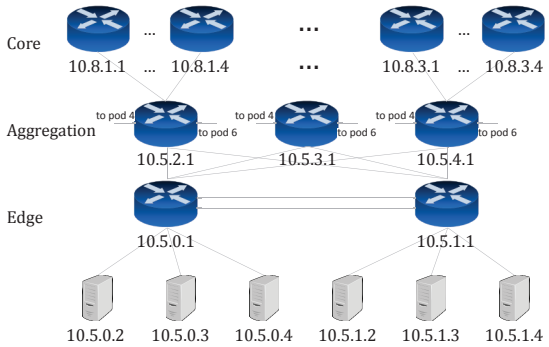


Fig. 7. Addressing of switches in pod 5 of a $(2, 2, 0, 1, 8)$ VCN.

IV. Routing Algorithms

In this section, we present the routing scheme for the VCN. Before getting into the routing algorithms, it is necessary to describe the addressing scheme. In the VCN, all the switches and the servers connected by the network are addressed under the $10.0.0.0/8$ prefix. The core switches are addressed by $10.k.x.y$, where k is the number of ports on each switch, $x = 1, 2, \ldots, \frac{k-h_i}{2}+i$, $y = 1, 2, \ldots, \frac{k-h_j}{2}+j$. In pod p, all the switches and the servers are addressed under the $10.p.0.0/16$ prefix, where

$$p = 0, 1, \ldots, k-1.$$

Specifically, the edge switches in pod p are addressed as $10.p.e.1$, where

$$e = 0, 1, \ldots, \frac{k-h_j}{2} - j - 1.$$

The aggregation switches in pod p are addressed as $10.p.a.1$, where

$$a = \frac{k-h_j}{2} - j, \ldots, k - \frac{h_i+h_j}{2} + i - j - 1.$$

For each edge switch $10.p.e.1$, all its connected servers are addressed under the $10.p.e.0/24$ prefix. That is, the servers are addressed from $10.p.e.2$ to $10.p.e.(\frac{k-h_i}{2} - i + 1)$. Fig. 7 shows an example of addressing in VCN, which includes the addresses of switches and corresponding servers in pod 5 of a $(2, 2, 0, 1, 8)$ VCN.

A. Facilitating Local Traffic Using Horizontal Links

During the design of our routing algorithms, the first principle we follow is to facilitate the routing of local traffic. We can consider again the example shown in Fig. 5 and Table II. Without horizontal links (Fig. 5(a)), the routing from server B to server D includes 4 hops (B-1-3-2-D, or B-1-4-2-D). Now because of the horizontal link between neighbor switches 1 and 2 (Fig. 5(b)), the routing from server B to server D could be either 4 hops (B-1-3-2-D) or reduced to 3 hops (B-1-2-D). In this case, local traffic (e.g., between server B and server D) should take the shorter route whenever possible (shown as the dashed lines). Only in the case of the shorter route is busy, local traffic then resorts to the longer route. In our example, local traffic between servers A and C takes longer route (shown as the solid lines), because the horizontal link is occupied by the traffic between servers B and D.

This principle also applies to more general cases. In the previous example, servers B and D are connected to different edge switches, or different /24 subnets. The routing can be shorten by horizontal links because edge switches 1 and 2 are neighbors. The same is true for the servers that are connected to different pods. Consider server $10.5.0.2$ in pod 5 as an example (Fig. 7). We assume that this server is sending data to another server $10.3.0.2$ (a destination in pod 3, according to our addressing scheme). The route must include 6 hops: 3 hops up to the core tier, and 3 hops down to the destination. However, if this server is sending data to another destination $10.4.0.2$ in pod 4, the route then can be shorter because pod 5 and pod 4 are neighbor pods. Specifically, the 3 aggregation switches $10.5.2.1$, $10.5.3.1$ and $10.5.4.1$ in pod 5 each has one horizontal link directly to pod 4. In this example, route between servers that belong to different /16 subnets can be shorten because of the horizontal links in the aggregation tier. To be more clear, we list the routing distance of local traffic for both cases in Table IV.

TABLE IV. Routing Distance of Local Traffic

traffic between	without horizontal links	via horizontal links
neighbor /24 subnets	4 hops	3 hops
neighbor /16 subnets	6 hops	5 hops

Guided by this principle, we propose a very concise routing strategy, which is shown as Algorithm 1. This routing strategy applies to every packet at every switch in the VCN.

In order to fully understand the routing strategy, we have two observations about the topology of the VCN, Observation 1: For any one packet at one switch, if there is a downward path from the switch to the packet's destination server, the downward path must be the shortest path from that switch to the destination server. This is true simply because it takes at least one hop to going down one tier. Therefore, a downward path should has highest priority (line 2 in Algorithm 1). Observation 2: A packet can always find a downward path at any one of the core switches. This observation holds because any one core switch has one downward link to each of the pods in the VCN, and any one aggregation switch has one downward link to each of the edge switches in its pod as well. Therefore, if the destination cannot be routed at current switch, forwarding it upwards can ultimately solve the situation (line 6 in Algorithm 1). But before the ultimate solution, we always try to route a packet locally via horizontal links (line 4 in Algorithm 1). In this way, the routing strategy follows the aforementioned principle of facilitating local traffic.

We enforce the routing strategy by encoding it into the

Algorithm 1 Routing Strategy of the VCN

```
1: if destination reachable via some down link then
2:     forward packet via the down link
3: else if destination reachable via some horizontal link then
4:     forward packet via the horizontal link
5: else
6:     forward packet upwards
7: end if
```

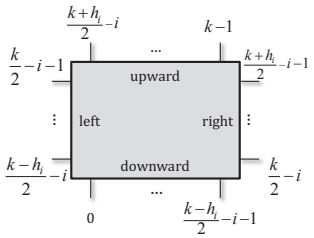
routing tables of all the switches in the VCN. Algorithms 2, 3, and 4 implement the automated routing configurations of all the edge switches, aggregation switches, and core switches respectively. These algorithms use a common function $addEntry()$ (e.g., line 5 in Algorithm 2), which takes three parameters. The first parameter specifies which switch an entry is added to its routing table, the second parameter specifies the destination server, and the third specifies the output port. Algorithms 2 and 3 literally follow the strategy in Algorithm 1, which include the handling of downward, horizontal, and upward traffic successively. Algorithm 4 only needs to handle the downward traffic, because core switches only have downward links. (Or in other words, because of our aforementioned Observation 2, the condition in line 1 of Algorithm 1 always evaluates to true). To be clear, we explicitly show three routing tables for three switches in Fig. 7, which are Table V for edge switch $10.5.0.1$, Table VI for aggregation switch $10.5.4.1$, and Table VII for core switch $10.8.3.1$. We can see that, the entries in Tables V and VI can be divided into three partitions, which handles downward, horizontal and upward traffic respectively. The entries in Tables VII only handle downward traffic.

In order to understand how local traffic is facilitated, we can take Table V for edge switch $10.5.0.1$ as an example. We also show the numbering of the ports in Fig. 8 so that the routing table is more readable. Entries in Table V are divided into three sections. When a packet is about to be forwarded, the three sections are checked in order. Once the packet hits some entry in one section, then the following section(s) are ignored. In this way, the routing strategy in Algorithm 1 is applied. For example, imagine a packet that has destination server $10.5.1.3$, which hits entries 5, 8 and 11. Since entry 11 is from the third section, it will be omitted. That is, the packet will not go upwards. Instead, it will be forwarded horizontally to the neighbor edge switch correctly. When there are multiple hits within one section of the routing table, the tie is broken by randomly select one entry. In this example, the packet hits entries 5 and 8. The reason why these two entries have the same destinations is that in our example VCN (Fig. 7), there are only two edge switches in each pod. Therefore, for edge switch $10.5.0.1$, its left neighbor and right neighbor are both edge switch $10.5.1.1$.

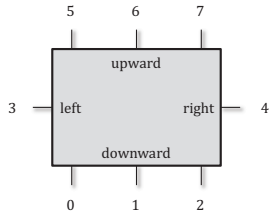
From the example of edge switch $10.5.0.1$, we can see that the horizontal links in the edge tier, along with Algorithm 2, altogether facilitate local traffic between neighbor /24 subnets. Similar to that, Algorithm 3 also use the horizontal links in the aggregation tier to facilitate local traffic between neighbor /16 subnets.

TABLE V. ROUTING TABLE OF EDGE SWITCH $10.5.0.1$ IN A $(2, 2, 0, 1, 8)$ VCN

entry index	destination	next hop	output
1	$10.5.0.2$	$10.5.0.2$	port 0
2	$10.5.0.3$	$10.5.0.3$	port 1
3	$10.5.0.4$	$10.5.0.4$	port 2
4	$10.5.1.2$	$10.5.1.1$	port 3
5	$10.5.1.3$	$10.5.1.1$	port 3
6	$10.5.1.4$	$10.5.1.1$	port 3
7	$10.5.1.2$	$10.5.1.1$	port 4
8	$10.5.1.3$	$10.5.1.1$	port 4
9	$10.5.1.4$	$10.5.1.1$	port 4
10	$10.X.X.2$	$10.5.2.1$	port 5
11	$10.X.X.3$	$10.5.3.1$	port 6
12	$10.X.X.4$	$10.5.4.1$	port 7



(a) (h_i, h_j, i, j, k) VCN



(b) $(2, 2, 0, 1, 8)$ VCN

Fig. 8. Port numbering of an edge switch in a VCN.

TABLE VI. ROUTING TABLE OF AGGREGATION SWITCH $10.5.4.1$ IN A $(2, 2, 0, 1, 8)$ VCN

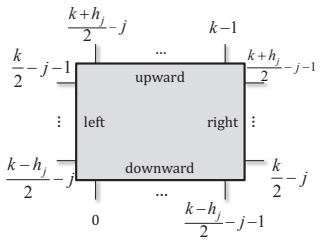
entry index	destination	next hop	output
1	$10.5.0.X$	$10.5.0.1$	port 0
2	$10.5.1.X$	$10.5.1.1$	port 1
3	$10.4.X.2$	$10.4.4.1$	port 2
4	$10.4.X.3$	$10.4.4.1$	port 2
5	$10.4.X.4$	$10.4.4.1$	port 2
6	$10.6.X.2$	$10.6.4.1$	port 3
7	$10.6.X.3$	$10.6.4.1$	port 3
8	$10.6.X.4$	$10.6.4.1$	port 3
9	$10.X.X.2$	$10.8.3.4$	port 7
10	$10.X.X.3$	$10.8.3.2$	port 4
11	$10.X.X.4$	$10.8.3.3$	port 5

TABLE VII. ROUTING TABLE OF CORE SWITCH $10.8.3.1$ IN A $(2, 2, 0, 1, 8)$ VCN

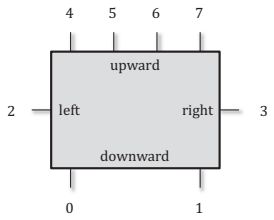
entry index	destination	next hop	output
1	$10.0.X.X$	$10.0.4.1$	port 0
2	$10.1.X.X$	$10.1.4.1$	port 1
3	$10.2.X.X$	$10.2.4.1$	port 2
4	$10.3.X.X$	$10.3.4.1$	port 3
5	$10.4.X.X$	$10.4.4.1$	port 4
6	$10.5.X.X$	$10.5.4.1$	port 5
7	$10.6.X.X$	$10.6.4.1$	port 6
8	$10.7.X.X$	$10.7.4.1$	port 7

Algorithm 2 Routing Configurations of Edge Switches for an (h_i, h_j, i, j, k) VCN

1: **for** pod $p \in [0, k-1]$ **do**
2: **for** edge switch $e \in [0, \frac{k-h_j}{2} - j - 1]$ **do**
3: //downward traffic
4: **for** server $s \in [2, \frac{k-h_i}{2} - i + 1]$ **do**
5: addEntry($10.p.e.1$, $10.p.e.s$, $s-2$);
6: **end for**
7: //horizontal traffic
8: $e_l = (e-1) \pmod{\left(\frac{k-h_j}{2} - j\right)}$; //left neighbor
9: **for** server $s \in [2, \frac{k-h_i}{2} - i + 1]$ **do**
10: $d = s \pmod{\frac{h_i}{2}} + \frac{k-h_i}{2} - i$;
11: addEntry($10.p.e.1$, $10.p.e_l.s$, d);
12: **end for**
13: $e_r = (e+1) \pmod{\left(\frac{k-h_j}{2} - j\right)}$; //right neighbor
14: **for** server $s \in [2, \frac{k-h_i}{2} - i + 1]$ **do**
15: $d = s \pmod{\frac{h_i}{2}} + \frac{k}{2} - i$;
16: addEntry($10.p.e.1$, $10.p.e_r.s$, d);
17: **end for**
18: //upward traffic, special case
19: **if** $i == 0$ **then**
20: **for** server $s \in [2, \frac{k-h_i}{2} + 1]$ **do**
21: $d = (s-2+e) \pmod{\frac{k-h_i}{2}} + \frac{k+h_i}{2}$;
22: addEntry($10.p.e.1$, $X.X.X.s$, d);
23: **end for**
24: continue;
25: **end if**
26: //upward traffic, general cases
27: **for** server $s \in [2, \frac{k-h_i}{2} - i + 1]$ **do**
28: $d = [s - 2 + (\frac{k-h_i}{2} - i)e]$;
29: $d = d \pmod{(\frac{k-h_i}{2} + i)} + \frac{k+h_i}{2} - i$;
30: addEntry($10.p.e.1$, $X.X.X.s$, d);
31: **end for**
32: **end for**
33: **end for**



(a) (h_i, h_j, i, j, k) VCN



(b) $(2, 2, 0, 1, 8)$ VCN

Fig. 9. Port numbering of an aggregation switch in a VCN.

Algorithm 3 Routing Configurations of Aggregation Switches for an (h_i, h_j, i, j, k) VCN

1: **for** pod $p \in [0, k-1]$ **do**
2: **for** aggr. switch $a \in [\frac{k-h_j}{2} - j, k - \frac{h_i+h_j}{2} + i - j - 1]$ **do**
3: //downward traffic
4: **for** edge switch $e \in [0, \frac{k-h_j}{2} - j - 1]$ **do**
5: addEntry($10.p.a.1$, $10.p.e.X$, e);
6: **end for**
7: //horizontal traffic
8: $p_l = (p-1) \pmod{k}$; //left neighbor
9: **for** server $s \in [2, \frac{k-h_i}{2} - i + 1]$ **do**
10: $d = s \pmod{\frac{h_j}{2}} + \frac{k-h_j}{2} - j$;
11: addEntry($10.p.a.1$, $10.p_l.X.s$, d);
12: **end for**
13: $p_r = (p+1) \pmod{k}$; //right neighbor
14: **for** server $s \in [2, \frac{k-h_i}{2} - i + 1]$ **do**
15: $d = s \pmod{\frac{h_j}{2}} + \frac{k}{2} - j$;
16: addEntry($10.p.a.1$, $10.p_r.X.s$, d);
17: **end for**
18: //upward traffic, special case
19: **if** $\frac{h_i}{2} + i == \frac{h_j}{2} - j$ **then**
20: **for** server $s \in [2, \frac{k-h_i}{2} - i + 1]$ **do**
21: $d = (s - 2 + p) \pmod{\frac{k-h_j}{2} + j} + \frac{k+h_j}{2} - j$;
22: addEntry($10.p.a.1$, $X.X.X.s$, d);
23: **end for**
24: continue;
25: **end if**
26: //upward traffic, general cases
27: **for** server $s \in [2, \frac{k-h_i}{2} - i + 1]$ **do**
28: $d = [s - 2 + (\frac{k-h_i}{2} - i)p]$;
29: $d = d \pmod{(\frac{k-h_j}{2} + j)} + \frac{k+h_j}{2} - j$;
30: addEntry($10.p.a.1$, $10.X.X.s$, d);
31: **end for**
32: **end for**
33: **end for**

Algorithm 4 Routing Configurations of Core Switches for an (h_i, h_j, i, j, k) VCN

1: **for** $x \in [1, \frac{k-h_i}{2} + i]$ **do**
2: **for** $y \in [1, \frac{k-h_j}{2} + j]$ **do**
3: **for** pod $p \in [0, k-1]$ **do**
4: addEntry($10.k.x.y$, $10.p.X.X$, p);
5: **end for**
6: **end for**
7: **end for**

B. Load Balancing

Aside from the facilitation of local traffic, our second consideration is load balancing. In our routing scheme, various designs are made to distribute the traffic evenly throughout the VCN.

Before going into details, we want to make it clear that the load balancing mechanisms are not applicable for a downward packet. Recall the Observation 1 about the topology: If there is a downward path to the destination server, the downward path must be the shortest path. Actually, this downward path is not only shortest, but also unique, because there is exactly one link from a switch down to an adjacent tier. In Algorithm 3, lines 3 to 6 handle the downward traffic. These lines of configurations generate entries 1 to 2 in Table VI. In order to make the table more readable, we also show the numbering

of the ports in Fig. 9. The functionality of entries 1 to 2 can be summarized as: For those packets which are destined for the servers in pod 5, they are distributed correctly down to the edge tier, based on their destination servers' addresses. In Algorithm 2, lines 3 to 6 handle the downward packets in a similar way.

Next, we discuss how load balancing is applied to horizontal and upward traffic in the VCN.

If the destination address of a packet indicates that the packet is destined for a server in a neighbor pod, then the packet is forwarded to an output port which connects to a horizontal link to that neighbor pod. In Algorithm 3, lines 7 to 17 handle the horizontal traffic. These lines of configurations generate entries 3 to 8 in Table VI. In our example, for simplicity, there is only one left port and one right port (Fig. 9(b)). If there are multiple left or right ports, these entries distribute packets evenly among these ports, based on the destination addresses of these packets. In Algorithm 2, lines 7 to 17 balance the horizontal traffic in the edge tier in a similar way.

If the destination address of a packet indicates that the packet is neither connected to the current pod nor connected to a neighbor pod, then it has to be forwarded upwards. In Algorithm 3, lines 18 to 31 handle the upward traffic. These line of configurations generate entries 9 to 11 in Table VI. There are two different load balancing mechanisms for the handling of upward traffic here. First, upward traffic is evenly distributed among the upward ports. Second, note that the output port is also related to p (line 28 in Algorithm 3). In this way, traffic that originates from different pod but are destined for the same server are distributed among different core switches. Therefore, the traffic from core tier to aggregation tier is also balanced. In short, the two mechanisms in Algorithm 3 not only balance the traffic from aggregation tier to the core tier, but also balance the traffic from core tier to the aggregation tier. In Algorithm 2, lines 18 to 31 provide load balancing in similar way.

Finally, routing tables for the core switches are straightforward, because 1) there are no horizontal links in the core tier, 2) as mentioned earlier, the load balancing from the core tier to the aggregation tier is already considered. Therefore, for each core switch, it simply forwards packets to the correct pods, as shown in Algorithm 4 and Table VII.

V. Profile Fitting of the VCN

With the horizontal connections deployed and the routing scheme applied, we are ready to introduce another unique feature of the VCN. In short, the VCN is able to fit its topology into the traffic profile running on it, so that we can compose the network infrastructure in a significantly more efficient way. We name this feature as *profile fitting* of the VCN.

As mentioned in Section I, the motivation of the design of VCN originates from traffic locality. In order to characterize the traffic locality, we use a triplet (t_{24}, t_{16}, t_8). t_{24} denotes the proportion of the traffic which is within the same /24 subnet. t_{16} denotes the proportion of the traffic which is within the same /16 subnet, but not within the same /24 subnet. t_8 denotes the proportion of the inter-pod traffic. We say (t_{24}, t_{16}, t_8) is the profile of the traffic, because it shows the distribution of the traffic based on its locality. Profile fitting takes the traffic profile as part of its input, and outputs a fitted topology. This fitted topology provisions sufficient bandwidth for all types of traffic, and simultaneously reduces the cost significantly.

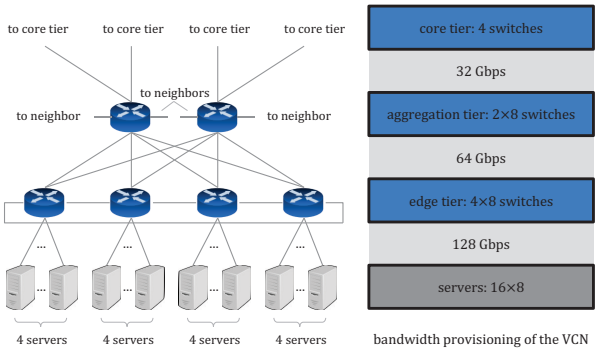


Fig. 10. An example of profile fitting. The traffic profile is (0.5, 0.25, 0.25). One of the fitted topologies could be a $(2, 2, -1, -1, 8)$ VCN. This figure shows one of the 8 pods, and the bandwidth provisioning of the entire VCN.

The fitting can be done easily as follows:

$$\begin{aligned} i &= -\frac{t_{24}}{2t_8 + 2t_{16} + t_{24}} \frac{k - h_i}{2}, \\ j &= -\frac{t_{16}}{2t_8 + t_{16}} \frac{k - h_j}{2}. \end{aligned} \tag{1}$$

If an (h_i, h_j, i, j, k) VCN satisfies (1), we say that its topology is fitted, in the sense that the inter-tier aggregate bandwidth provisioning is tailored according to the traffic profile. Assume that we have a traffic profile $(0.5, 0.25, 0.25)$. Using 8-port GigE switches to fit the profile, one possible result is a (2,2,-1,-1,8) VCN, which is shown as Fig. 10. From the profile, we can see that $t_{24} = 0.5$, which means half of the traffic is within the same /24 subnet. In other words, half of the traffic can be routed using edge switches only. Correspondingly, the bandwidth provisioning between the edge tier and the aggregation tier (64 Gbps) is a half of that between the edge tier and the servers (128 Gbps). Also, we can see that $t_8 = 0.25$, which means only a quarter of the traffic needs to be routed using the core switches. Therefore the bandwidth provisioning to the core tier (32 Gbps) is one fourth of that to the servers (128 Gbps). We can conveniently scale up the fitted VCN by choosing a different k. For example, using 48-port GigE switches, we can have a $(12, 12, -6, -6, 48)$ VCN, which also fits the same traffic profile $(0.5, 0.25, 0.25)$. The $(12, 12, -6, -6, 48)$ VCN provides GigE links to 27,648 servers.

The benefit of profile fitting is that the network devices are used more efficiently by the VCN. We can see this clearly by comparing the VCN with the fat-tree. In the aforementioned $(12, 12, -6, -6, 48)$ VCN, we use a total of 1,872 switches to provide connections to 27,648 servers. Given that the 48-port GigE switches for data centers (e.g., Cisco Nexus 3000 Series) are priced at around 3,000 US dollars, the total cost on the switches is 5.62 millions, or \$0.2K/Gbps per server. On the other hand, a 48-ary fat-tree uses 2,880 switches to connect the same number of servers, which result in a total cost of 8.64 millions, or \$0.3K/Gbps per server. This cost is about 50% higher than that of the VCN.

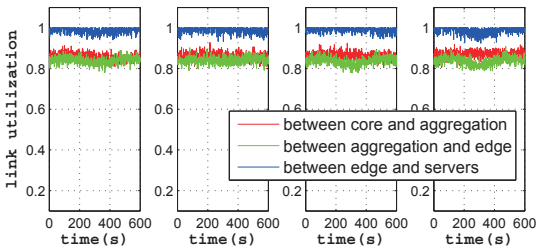
The reason of the cost cut in VCN is simply that over provisioned devices are removed from the fabric (recall the intuition shown as Fig. 2). Take a close look at the bandwidth provisioning of the fat-tree, we can see that the inter-tier aggregate bandwidth is a constant for different tiers. With the ubiquitous presence of traffic locality, this provisioning would be overkill. Because a k-ary fat-tree can be see as a special $(0, 0, 0, 0, k)$ VCN, which corresponds to a traffic profile of $(0, 0, 1)$. This profile indicates that all traffic are inter-pod traffic and must be routed by core switches, which, as mentioned in Section I, is not the practical case. Using profile fitting, idle devices are removed from the network, leaving those which are essential for the real workload. Again, we can safely remove them because we have the horizontal links deployed and the routing scheme applied, which provide sufficient cushion against traffic fluctuations.

VI. Performance Evaluations

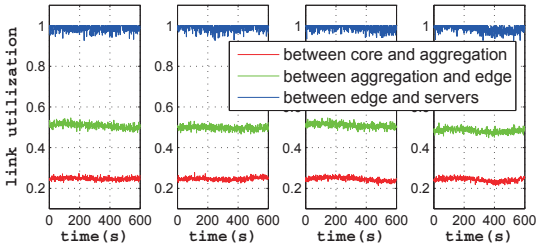
In this section we present various performance evaluations of the VCN. Our platform is Ubuntu 16.04 LTS running on an Amazon EC2 VM with 8 cores and 64 GB memory. On this platform, we simulate a $(2, 2, -1, -1, 8)$ VCN data center and an 8-ary fat-tree data center, and make comparisons between them. The simulated data centers are setup in three steps as follows. First, we build the networks using ns-3 [26], which is a powerful discrete-event network simulator. Second, we run the servers with Linux Containers (LXC) [27], an operating-system-level, light-weighted, virtualization environment. Finally, we connect the servers to the networks using TapBridge NetDevice, which is an integrated facility coming with ns-3.

With the simulated data centers, we first implement the routing scheme in Section IV, by generating routing table entries and adding the entries to the simulated switches. For fat-tree, we use its original routing algorithm, which is described in [3]. Next, we simulate the traffic profile, by implementing traffic generators and deploying them on all the servers. A traffic generator simply runs in an everlasting loop, and continuously generates packets to other servers. The destination address and size of each packet are randomized, but the distribution of the destinations follows the traffic profile. In order to demonstrate the performance of the VCN, all traffic generators keep increasing their data rates until the capacities of their links to the edge switches are depleted.

We find the utilizations of various types of links in the VCN, and compare the utilizations with that of the fat-tree. The results are shown in Fig. 11. We randomly select three vertical links, one between the core tier and the aggregation tier, on between the aggregation tier and the edge tier, and one between the edge tier and the servers. The links between the edge tier and the servers are almost 100% busy, because of our settings for the traffic generators. The other two, as shown in Fig. 11(a), are well controlled: The utilization is kept at around 80%, but does not reach 100%, which means the network resources are efficiently used, and simultaneously congestions are avoided. We randomly select four set of the three links, all of them behave similarly. This is the result of both our accurate profile fitting and our combination of the horizontal links and the routing scheme. In the 8-ary fat-tree, the links between the servers and the edge tier are also

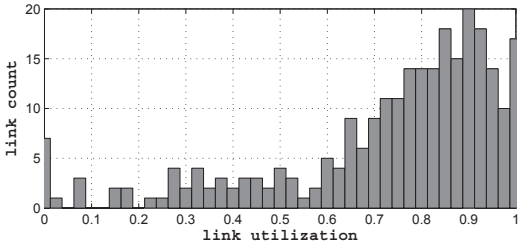


(a) $(2, 2, -1, -1, 8)$ VCN

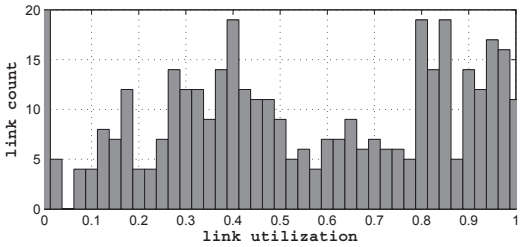


(b) 8-ary fat-tree

Fig. 11. Utilization of vertical links in a $(2, 2, -1, -1, 8)$ VCN and an 8-ary fat-tree.



(a) $(2, 2, -1, -1, 8)$ VCN

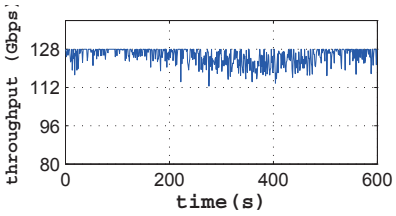


(b) 8-ary fat-tree

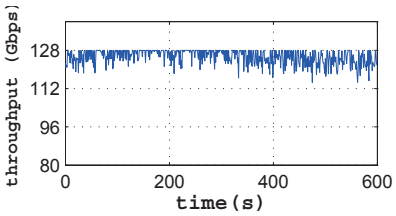
Fig. 12. Distribution of the utilization of links in a $(2, 2, -1, -1, 8)$ VCN and an 8-ary fat-tree.

100% busy because of our experimental settings, but the other two types of links are usually under-utilized, as shown in Fig. 11(b). This is because of the over-provisioning in the context of traffic locality.

In order to further understand the traffic in the networks, we visualize the distribution of the link utilizations in both the $(2, 2, -1, -1, 8)$ VCN and the 8-ary fat-tree. Fig. 12(a) shows the distribution of the link utilizations in the VCN. We can see that a majority of the links have a relative high utilization, comparing to that of the 8-ary fat-tree in fat-tree (Fig. 12(b)). In Fig. 12(b), we can see that a significant amount of links are under-utilized, for example, there is a peak in the vicinity of utilization around 0.4. This result is in accordance with the observations that we obtained from Fig. 11. Therefore,



(a) $(2, 2, -1, -1, 8)$ VCN



(b) 8-ary fat-tree

Fig. 13. Throughput of a $(2, 2, -1, -1, 8)$ VCN and an 8-ary fat-tree.

the $(2, 2, -1, -1, 8)$ VCN indeed uses its network resources efficiently. (The second challenge in Section I-D resolved.)

Finally, we find the overall throughput for both the two networks and show them as Fig. 13. We can see that both the $(2, 2, -1, -1, 8)$ VCN and the 8-ary fat-tree are capable to achieve a full aggregate bandwidth (128 Gbps) of the servers. In other words, although we keep relatively high link utilizations, the traffic is still running fluently because we have the horizontal links and elaborately designed routing scheme. As a result, we managed to avoid the performance degradation from possible traffic congestions. (The first challenge in Section I-D resolved.)

From these simulations, we can see that the $(2, 2, -1, -1, 8)$ VCN delivers the same performance as the fat-tree, but at a significantly reduced hardware cost.

VII. Conclusions

In this work, we proposed VCN, a Clos-type network architecture for data centers which takes the traffic locality into consideration. First, we introduced horizontal connections into the topology. We also proposed the addressing and routing schemes that utilize the horizontal connections to facilitate local traffic and balance the workload among all the network devices. With these techniques as safety cushions against traffic fluctuations, we can fit the topology of the VCN into the profile of the localized traffic, so that the network devices in the topology are used more efficiently. In this way, the VCN can deliver the same throughput as its predecessors. Meanwhile, with the fitted topology and efficiently used network devices, the cost of the network infrastructure can be sharply reduced. Finally, the VCN can be seen as a generalization of fat-tree, making it ready to be deployed in the widely adopted fat-tree data centers.

ACKNOWLEDGMENTS

This research work was supported in part by the U.S. National Science Foundation under grant numbers CCF-1320044 and CCF-1717731.

References

[1] C. Clos, "A study of non-blocking switching networks," *Bell Syst. Tech. J.,* vol. 32, no. 2, pp. 406-424, 1953.

[2] C. E. Leiserson, "Fat-trees: universal networks for hardware-efficient supercomputing," *Computers, IEEE Trans.,* vol. 100, no. 10, pp. 892-901, 1985.

[3] M. Al-Fares, A. Loukissas and A. Vahdat, "A scalable, commodity data center network architecture," *ACM SIGCOMM Comput. Commun. Rev.,* vol. 38, no. 4, pp. 63-74, 2008.

[4] A. Greenberg et al., "VL2: a scalable and flexible data center network," *ACM SIGCOMM Comput. Commun. Rev.,* vol. 39, no. 4, 2009.
"Oversubscription bounded multicast scheduling in fat-tree data center networks",

[5] Z. Guo, J. Duan, and Y. Yang, "On-line multicast scheduling with bounded congestion in fat-tree data center networks," *IEEE JSAC* vol. 32, no. 1, pp. 102-115, 2014.

[6] A. Roy et al., "Inside the Social Network's (Datacenter) Network," *ACM SIGCOMM Comput. Commun. Rev.,* vol. 45, no. 4, pp. 123-137, 2015.

[7] A. Singh et al., "Jupiter Rising: A Decade of Clos Topologies and Centralized Control in Google's Datacenter Network," *ACM SIGCOMM Comput. Commun. Rev.,* vol. 45, no. 4, pp. 183-197, 2015.

[8] Cisco Data Center Infrastructure 2.5 Design Guide, [Online]. Available: http://www.cisco.com

[9] N. Farrington and A. Andreyev, "Facebook's data center network architecture" *Proc. Optical Interconnects Conf,* pp. 49-50, 2013.

[10] Apache Hadoop, [Online]. Available: http://hadoop.apache.org/

[11] M. Hammoud and M. F. Sakr, "Locality-aware reduce task scheduling for MapReduce," *Cloud Computing Technology and Science (CloudCom), 2011 IEEE Third International Conference on,* IEEE, 2011.

[12] Y. Yang and G.M. Masson, "Nonblocking broadcast switching networks," *IEEE Trans. Computers*, vol. 40, no. 9, pp. 1005-1015, 1991.

[13] Y. Yang and G.M. Masson, "The necessary conditions for Clos-type nonblocking multicast networks," *IEEE Trans. Computers*, vol. 48, no. 11, pp. 1241-1227, 1999.

[14] Y. Yang and J. Wang, "On blocking probabilty of multicast networks." *IEEE Trans. Communications*, vol. 46, no. 7, pp. 957-968, 1998.

[15] W. Wang et al., "Maptask scheduling in mapreduce with data locality: Throughput and heavy-traffic optimality," *IEEE/ACM Transactions on Networking,* vol. 24, no. 1, pp. 190-203, 2016.

[16] K. Wang et al., "Load-balanced and locality-aware scheduling for data-intensive workloads at extreme scales," *Concurrency and Computation: Practice and Experience*, vol. 28, no. 1, pp. 70-94, 2016.

[17] X. Meng, V. Pappas, and L. Zhang, "Improving the scalability of data center networks with traffic-aware virtual machine placement," *INFOCOM, 2010 Proceedings IEEE,* 2010.

[18] X. Li, et al., "Let's stay together: Towards traffic aware virtual machine placement in data centers," *INFOCOM, 2014 Proceedings IEEE,* 2014.

[19] W. Fang, et al., "VMPlanner: Optimizing virtual machine placement and traffic flow routing to reduce network power costs in cloud data centers," *Computer Networks,* vol. 57, no. 1, pp. 179-196, 2013.

[20] X. Qin and Y. Yang, "Multicast connection capability of WDM swithing networks with limited wavelength conversion," *IEEE/ACM Trans. Networking*, vol. 12, no. 3, pp. 526-538, 2004.

[21] Y. Yang and G.M. Masson, "Broadcast ring sandwich networks," *IEEE Trans. Computers*, vol. 44, no. 10, pp. 1169-1180, 2004.

[22] Y. Yang and J. Wang, Optimal all-to-all personalized exchange in a class of optical multistage networks," *IEEE Transactions on Parallel and Distributed Systsems*, vol. 12, no. 6, pp. 567-582, 2001.

[23] J. Piao and J. Yan, "A network-aware virtual machine placement and migration approach in cloud computing," *Grid and Cooperative Computing (GCC), 2010 9th International Conference on, IEEE*, 2010.

[24] A. Andreyev, "Introducing data center fabric, the next-generation Facebook data center network," https://code.facebook.com/posts/360346274145943, 2014.

[25] C. D. Galler, "A Tour of Cisco's Allen Data Center," http://blogs.cisco.com/perspectives/a-tour-of-ciscos-allen-data-center

[26] ns-3 network simulator, [Online]. Available: https://www.nsnam.org/

[27] Linux Containers, [Online]. Available: https://linuxcontainers.org/