Diamond: An Improved Fat-tree Architecture for Large-scale Data Centers

Yantao Sun, Jing Chen, Qiang Liu, and Weiwei Fang
School of Computer and Information Technology, Beijing Jiaotong University, Beijng 100044, China
Email: ytsun@bjtu.edu.cn; yourney3@gmail.com, liuq@bjtu.edu.cn, wwfang@bjtu.edu.cn

Abstract—With the widespread deployment of cloud services, data center networks are developing toward large-scale, multi-path networks. To support the new trend, some novel network architectures have been proposed, and Fat-tree is one of the most promising architecture and gets a lot of attention because it has good performance on aggregate bandwidth with a simple topology. This paper presents Diamond, an improved Fat-tree architecture for large-scale data centers. Diamond replaces all the aggregation switches of Fat-tree with edge switches and connects directly edge switches to core switches. By this alteration, the average route path and End-to-End (ETE) delay in Diamond are 10% less than that in Fat-tree. We design FAR, a simple and high-efficient routing method for Diamond. In FAR, each switch requires only hundreds of route entries to support a large-scale network with tens of thousands servers, and to build FAR routing tables, switches exchange very few messages. Diamond and its FAR routing are verified through OPNET simulations.

Index Terms—data center network, network architecture, routing method

I. Introduction

In recent years, with the rapid development of cloud computing technologies, the widely deployed cloud services, such as Amazon EC2 and Google search, bring about huge challenges to data center networking (DCN)[1]. Today's data centers (DCs) require large-scale networks with higher internal bandwidth and lower transfer delay, but conventional networks cannot meet such requirements due to limitations in their network architecture.

Layered multi-root tree architecture is commonly used in conventional DCs, where many layer-2 domains connect together via layer-3 networks, as shown in Fig. 1. A layer-2 domain consists of many servers, which are divided into subnets, i.e., Virtual Local Area Network (VLANs), and each subnet contains up to a few hundred servers connected via switches.

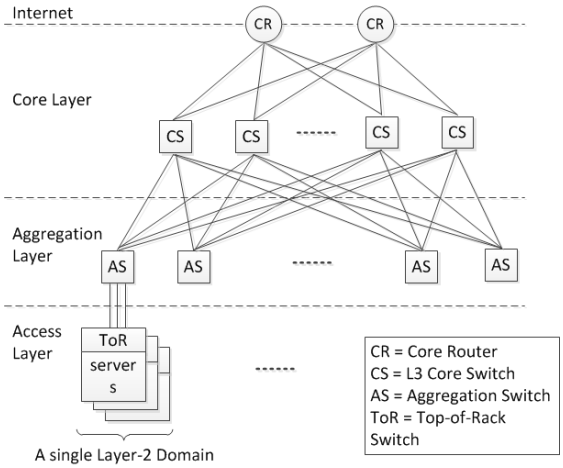


Fig. 1. A conventional network for data centers.

The architecture mentioned above cannot support a large-scale DC with up to tens of thousands of servers in one site. The main problem that limits a DCN's scale is the shortage of bandwidth in higher layers. In practice, a typical bandwidth oversubscription ratio between neighbor layers is 1:5 or more, and at the top layer, it might reach 1:80 to 1:240 [1]. Although the most advanced and fastest switches or routers are adopted in this architecture, only 50% of the aggregate bandwidth of edge networks can be supported by the top layer [2]. It is no doubt that the top layer is becoming a bottleneck for the entire network, especially in a cloud computing environment, when the requirement for intra-network traffic is increasing rapidly.

To solve the problems above, some new network architectures are proposed, such as Fat-tree [3], BCube [4], MatrixDCN [5], PortLand [6] and etc. Among those architectures, Fat-tree is the most promising architecture with many advantages: 1) it can support a large-scale network with 27,648 servers by using 2,880 switches; 2) it supports rearrangeable non-blocking routing because its bandwidth oversubscription ratio between neighbor layers can reach 1:1; 3) its topology is very simple and is similar to the conventional multi-root tree structure, which makes it easier to upgrade traditional switches to support Fat-tree architecture.

Many researches are carried out focusing on Fat-tree architecture. The paper [7] proposed a generic and automatic address configuration method for Fat-tree. PortLand [6] proposed a scalable, fault-tolerant layer-2 routing and forwarding protocol for Fat-tree. VL2 [1],

Manuscript received June 30, 2013; revised August 29, 2013.

In this paper, depending on the context, DCN represents data center network and data center networking interchangeably.

This work was supported by the ZTE-BJTU Collaborative Research Program under Grant No. K11L00190 and the Chinese Fundamental Research Funds for the Central Universities under Grant No. K12JB00060.

Corresponding author email: ytsun@bjtu.edu.cn

doi:10.12720/jcm.9.1.91-98

Helios [8], and c-Through [9] replace cheap core and aggregation switches with expensive, high-speed switches to reduce links in Fat-tree networks. In practice, Alibaba Company built a large cloud DC that can support concurrent 200Mbps traffics of 12 thousands of servers.

Fat-tree is becoming the most mainstream DCN architecture. In this paper, we propose Diamond, an improved Fat-tree architecture. In Diamond, the three-layer structure of Fat-tree is changed to two conjunct two-layer structures, and the aggregation switches in Fat-tree are replaced by edge switches. In Diamond, Each edge switch is directly connected to core switches, so a route path among different pods is shorter than that in Fat-tree.

Combining with the regularity in topology, we import a new routing method named Fault-avoidance Routing (FAR) for Diamond. FAR routes packets by means of two routing tables, a basic routing table (BRT) and a negative routing table (NRT). By looking up the BRT, FAR obtains a set of candidate routes, and by looking up the NRT, FAR obtains a set of avoiding routes. The final routes are the candidate routes minus the avoiding routes.

Compared with Fat-tree and other network architectures, Diamond has some advantages:

- Diamond can support large-scale DCNs as same as the Fat-tree. Using 48-port switches, a large-scale DCN can be built to accommodate up to 27,648 servers.
- A diamond network has multiple paths between a pair of severs, which insure the network's reliability and robustness.
- Diamond reduces the path length of routes. The average path length of routes in Diamond is 4.5 hops, compared with 5 hops in Fat-tree. Correspondingly, the end-to-end (ETE) delay in Diamond is about 10% less than that in Fat-tree.
- The distribution of traffic in Diamond is more even than that in Fat-tree, so Diamond has a lower congestion probability than Fat-tree.
- FAR routing is more efficient and has less computing cost than the routing of Fat-tree.

The main contributions of this paper are three-fold: 1) propose an improved Fat-tree architecture for large-scale DCNs; 2) design a new high-efficient routing method for Diamond; 3) achieve a thorough simulation study of Diamond with OPNET.

The remainder of this paper is organized as follows. Section 2 introduces the related research for DCN architecture. Section 3 presents the network fabric and addressing of Diamond, and gives an analysis of its advantages. Section 4 designs the FAR routing method for Diamond. Section 5 evaluates the Diamond's performance, including aggregate throughput and ETE delay, through OPNET simulations. Section 6 concludes for this paper.

II. Related Work

To solve the bandwidth bottleneck and network scalability issues, many novel network architectures for DCNs have been proposed over the past several years. Usually, these network structures can be divided into 3 types, switch-centric networks, server-centric networks and irregular networks.

Fat-tree [3] is a typical switch-centric network. In 2008, M. Al-Fares et al. proposed the Fat-tree network on SIGCOMM. Fat-tree is a rearrangeable non-blocking multi-path network in which there are many equal-cost paths between adjacent layers. It eliminates the bandwidth bottleneck in the core layer. Furthermore, Fat-tree can support large-scale networks with tens of thousands of physical servers. Using 48-port switches, a Fat-tree network can accommodate up to 27,648 servers. To support non-blocking communication, Fat-tree requires a large number of switches in the core and aggregation layers and wires to interconnect those switches, which increase deployment cost, energy consumption and management complexity. To solve the problem above, VL2 [1], Helios [8] and c-Through [9] replace cheap core and aggregation switches with expensive high-speed electrical switches and optical circuit switches. HyScale [10] is a non-tree structure that has high scalability. It uses hybrid optical networks, in which optical burst switching and optical circuit switching technologies are applied to transmit low and high volumes of data, respectively.

DCell [11] and BCube [4] are two types of server-centric networks that were proposed by Microsoft Research Asia (MSRA). DCell uses servers with multiple network ports and mini-switches to construct its recursively defined architecture. The weakness of DCell is its low bisection bandwidth, which may lead to traffic jams in its network. To solve traffic jams, MSRA proposed BCube, which provides more bandwidth in the top layer. In BCube, multi-port servers connect with multiple switches deployed in each layer. DCell and BCube can support extra-large-scale networks with more than 100 thousands of servers. FiConn [12] shares the same design principle as DCell. In contrast to DCell, a server node's degree in a k-level DCell is $k + 1$, FiConn always limits a server node's degree to two. In DPillar [13], the network is built of two types of devices, n-port switches and dual-port servers. Servers and Switches are arranged into k columns, respectively. Visually, it looks like the 2k columns of servers and switches are attached to the cylindrical surface of a pillar. In 2011, Deke Guo et al. proposed an expansible network structure for data centers using hierarchical compound graphs (BCN) [14]. Compared with the structures above, BCN is more complicated; and similar to DCell, it cannot eliminate the network's traffic bottleneck.

Some studies focus on irregular topologies. Scafida [15] is inspired by the scale-free Barabási and Albert topologies. Curtis et al. proposed REWIRE [16], which is

a framework to design, upgrade, and expand data center networks. REWIRE uses a local search to find a network that maximizes the bisection bandwidth while minimizing the latency and satisfying a group of user-defined constraints.

There are still some other research studies on network architectures. In 2007, Microsoft proposed using standard shipping containers to modularize data centers [17]. The uFix [18] designed a scalable and modularized architecture to interconnect heterogeneous data center containers, in which each container can have a different structure, such as Fat-tree or BCube. Wireless technology is also used in data center networks. In 2011, Daniel Halperin et al. proposed adding multi-gigabit wireless links to provide additional bandwidth [19].

In practice, most of today's data centers are based on switch-centric architectures. Although their scalability and flexibility is not good enough, switch-centric architectures have innate advantages; they are similar to traditional network architecture, which makes it easier to upgrade traditional switches to support them. Most of the network components and protocols can be used in switch-centric architectures directly or with a slight modification.

III. Diamond Architecture

In this section, we reform the Fat-tree architecture and propose a symmetrical Fat-tree architecture. This architecture looks like a diamond, so we call it Diamond architecture.

A. Network Fabric

We build a Diamond network using layer-3 switches. There are two types of switches in Diamond, i.e. core switches and edge switches. We suppose a switch has k ports. All the edge switches are divided into k pods and each pod has k edge switches. There are total k^2 edge switches. In a pod, edge switches are deployed as two lines and each line has k^2 edge switches. All the core switches are deployed as two lines. The core switches in the upper line connect edge switches of upper line in each pod together. The core switches in the lower line connect edge switches of lower line in each pod together. All the edge switches, which lie at the same position in each pod, connect to the same core switch. All the servers are connected to edge switches. Fig. 2 is an example of a diamond network with 4-port switches.

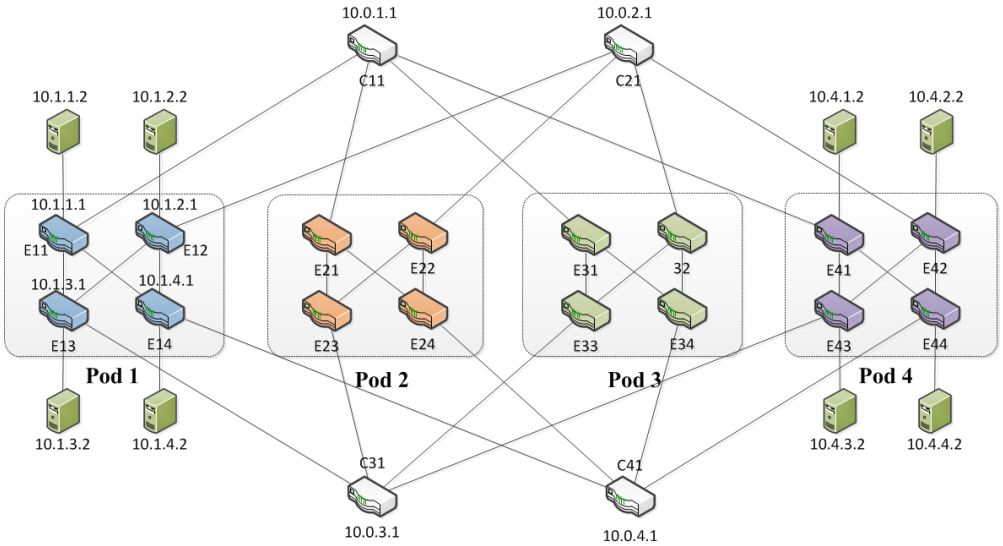


Fig. 2. A Diamond network with 4-port switches

For the k ports of an edge switch, $k/2$ ports connect with all the edge switches in opposite line in the same pod, $k/4$ ports connect with core switches, and the remaining $k/4$ ports connect with servers. There are total $k/4 \times k = k^2/4$ core switches and $k/4 \times k \times k = k^3/4$ servers in the entire network.

B. Addressing

Switches and servers are addressed according to their location in the network. We use E_{ij} to denote an edge switch whose IP address is 10.i.j.1 where i denotes the pod number (in [1, k]), j denotes the position of the edge switch in the pod (in [1, k], starting from left to right, top to bottom). C_{ij} denotes a core switch whose IP address is 10.0.i.j where i denotes that the core switch connect with the set of edge switches $\{E_{xi} \mid x = 1,2,..., k, i \in [1, k]\}$, and j denotes the switch number (in [1, $k/4$]) in the set of core switches that connect with the set E_{xi}. A server connected to E_{ij} is given an address in the form of 10.i.j.x where x is the server number (in [2, $k/4 + 1$].

C. Average Path Length

The differences between Diamond and Fat-tree are: 1) all the aggregation switches in Fat-tree are replaced by edge switches in Diamond; 2) all the edge switches are directly connected to core switches in Diamond. After this alteration, the distribution of servers in Diamond is more even than that in Fat-tree, as a result, the average path length of inter- or intra-pod routes in Diamond is shorter than that in Fat-tree and the ETE delay of a traffic

flow in Diamond is smaller than that in Fat-tree too. The average path length in Diamond and Fat-tree is calculated and listed in Table. I.

TABLE I: AVERAGE PATH LENGTH IN DIAMOND AND FAT-TREE

Network Type	Intra-pod Path	Inter-pod Path
Diamond	$\frac{5k^2 - 4k - 8}{2k^2 + 8} \approx 2.5$	$\frac{9k - 2}{2k} \approx 4.5$
Fat-tree	$\frac{3k + 2}{k + 2} \approx 3$	5

Table I shows that the average path length in Diamond is 0.5 hops shorter than that in Fat-tree, correspondingly, the ETE delay in Diamond will be about 10% smaller than that in Fat-tree. In DCNs, with the CPU speed of servers growing up, the network bandwidth and transmission delay is becoming the bottleneck of DCN performance, so reducing transmission delay helps to improve DCN's Performance. Moreover, the up-link traffic of each edge switch is directly forwarded to a core switch, so the traffic distribution is even on core switches, which lessens the congestion on core switch, and increases the aggregate throughput of the network. Simulation results in Section 5 prove the conclusions above.

IV. FAR ROUTING METHOD

A fault-avoidance routing (FAR) method is used in Diamond Routing. By leveraging the regularity in the topology of Diamond, FAR implements a simple, effective and fault-tolerant routing algorithm. Different with other routing methods, FAR uses two routing tables, a basic routing table (BRT) and a negative routing table (NRT). The function of a BRT is the same as that of a routing table in conventional routing methods, telling FAR what paths are available to reach the destination node. Oppositely, NRT tells what paths FAR should avoid because those paths pass through some failed links. FAR looks up the two routing tables to calculate the final applicable paths.

A. Basic Routing Table

By leveraging the regularity in topology, FAR can calculate routing paths for any destination and build a BRT for a router, without the complete knowledge of a network topology.

1) BRT of edge switches

We take E_{11} as an example to introduce how to build a basic routing table for an edge switch.

(a) Intra-pod Routing. For intra-pod routing, the destination and E_{11} lie in the same pod. There are two types of routes in this case.

- The destination connects to E_{11}. E_{11} can forward incoming packets to the destination through layer-2 switching, so no route entries are required.
- The destination connects to an edge switch in opposite line. E_{11} will forward incoming packets to the corresponding edge switch in opposite line. The required route entries are:

Destination/Mask	Next Hop
10.1.3.0/255.255.255.0	10.1.3.1
10.1.4.0/255.255.255.0	10.1.4.1

- The destination connects to an edge switch in the same line. E_{11} will forward incoming packets to any of edge switches in opposite line at first, and then the switch in opposite line forwards the packet to the destination switch. The required route entries are:

Destination/Mask	Next Hop
10.1.0.0/255.255.0.0	10.1.3.1
10.1.0.0/255.255.0.0	10.1.4.1

(b) Inter-pod Routing. For inter-pod routing, the destination and E_{11} lie in different pods. There are 3 types of routes that E_{11} forwards incoming packet to the destination.

- The routes that pass through the core switches that E_{11} directly connect to. The required route entries are (Type 1):

Destination/Mask	Next Hop
10.0.0.0/255.0.0.0	10.0.1.1

- The routes that pass through the edge switches in opposite line. The required route entries are (Type 2):

Destination/Mask	Next Hop
10.0.0.0/255.0.0.0	10.1.3.1
10.0.0.0/255.0.0.0	10.1.4.1

- For some destination, such as Node 10.4.3.2, the route passing through E_{13} is 2 hops longer than a route passing through E_{14} in the routes of type 2, so the route passing through E_{13} should have priority. To solve the problem, some route entries are required (Type 3):

Destination/Mask	Next Hop
10.0.3.0/255.0.255.0	10.1.3.1
10.0.4.0/255.0.255.0	10.1.4.1

All the routes above compose the BRT of E_{11}. In inter-pod routing, the path length of type 1 is equal to or shorter than that of type 2 and 3, so type 1 has the highest priority, i.e. the priority is: *type* 1 > *type* 3 > *type* 2. The priority can be set in their path metric.

2) BRT of core switches

Core switches are responsible for forwarding incoming packets among pods. The BRT of a core switch is very simple. Each pod has a corresponding route entry in a BRT of each core switch. For example, the BRT of C_{11} is composed of route entries:

Destination/Mask	Next Hop
10.1.0.0/255.255.0.0	10.1.1.1
10.2.0.0/255.255.0.0	10.2.1.1
10.3.0.0/255.255.0.0	10.3.1.1
10.4.0.0/255.255.0.0	10.4.1.1

B. Negative Routing Table

A BRT is calculated without considering link or switch failures in the entire network, so a switch cannot select proper routes to avoid network failures. To achieve fault-tolerance, another routing table, i.e. NRT, is used in FAR. FAR uses a NRT to exclude routes that pass through some failed links from the paths calculated by a BRT.

In a Diamond network, only edge switches require NRTs. NRT entries are related to the location of link failures. We take the switch E_{11} as an example to illustrate the procedure of building a NRT and discuss it for various link failures.

- A link that attaches to E_{11} fails, such as link $E_{11}\leftrightarrow C_{11}$ or $E_{11}\leftrightarrow E_{13}$. In this case, just remove routes related to the failed link from the BRT of E_{11}.
- A link that lies in the same pod with E_{11} but doesn't attach to E_{11} fails, such as link $E_{12}\leftrightarrow E_{13}$. In this case, to avoid the failed link, the route from E_{11} to E_{12} should avoid node E_{13}. We can do it by adding a route entry to the NRT of E_{11}:

Destination/Mask	Next Hop
10.1.2.0/255.255.255.0	10.1.3.1

The route above means that E_{11} should avoid 10.1.3.1 if it forwards a packet to the destination subnet 10.1.2.0/255.255.255.0.

- A link in other pod fails, such as link $E_{41}\leftrightarrow E_{43}$. In this case, the routes to E_{43} should avoid E_{41} and the routes to E_{41} should avoid E_{43}. We can do it by adding route entries to the NRT of E_{11}:

Destination/Mask	Next Hop
10.4.1.0/255.255.255.0	10.1.3.1
10.4.3.0/255.255.255.0	10.0.1.1

- A link of a core switch fails, such as link $C_{11}\leftrightarrow E_{21}$, and the core switch C_{11} connects with E_{11} and the failed link doesn't attach to E_{11}. In this case, the routes to pod 2 should avoid the core switch C_{11}. We can do it by adding route entries to the NRT of E_{11}:

Destination/Mask	Next Hop
10.2.0.0/255.255.0.0	10.0.1.1

- A link of a core switch fails, and the core switch, such as C_{21}, lies at the same side with E_{11} but doesn't connect to E_{11}. Because no routes of E_{11} go through C_{21}, the link failure has no effect on the routing of E_{11}.
- A link of a core switch fails, and the core switch, such as C_{31}, lies at opposite side of E_{11}. If link $E_{13}\leftrightarrow C_{31}$ fails, C_{31} can't forward any inter-pod packets for E_{11}, then we need to add a route entry to the NRT of E_{11} to avoid E_{13}:

Destination/Mask	Next Hop
10.0.0.0/255.0.0.0	10.1.3.1

If link $C_{31}\leftrightarrow E_{23}$ fails, C_{31} can't forward inter-pod packets to pod 2 for E_{11}, and then we need to add a route entry to the NRT of E_{11}:

Destination/Mask	Next Hop
10.2.0.0/255.255.0.0	10.1.3.1

We have discussed the effect of various link failures on the NRT of an edge switch. Because a node failure is equal to a group of link failures, we don't discuss node failure here.

C. Routing Procedure

FAR looks up both a BRT and a NRT to decide the next hop for a forwarding packet.

First, FAR takes the destination address of the forwarding packet as a criterion to look up route entries in a BRT based on longest prefix match. All the matched entries are composed of a set of candidate entries. The prefix length of a route is decided by the route's destination mask. Prefix length is 24, 16 and 8 for the mask 255.255.255.0, 255.255.0.0, and 255.0.0.0, respectively, and we define a route's prefix length is 8 for the discontinuous mask 255.0.255.0.

Second, FAR looks up route entries in an NRT also taking the destination address as criteria. In this lookup, no regard to longest prefix match, any entry that matches the criteria would be matched and composed of a set of avoiding entries.

At last, the candidate entries minus the avoiding entries compose the set of applicable entries. FAR sends the forwarding packet to anyone of the applicable entries. But for the inter-pod routing, FAR should select a route according to their priority. If the set of applicable entries is empty, the forwarding packet will be dropped.

We take the following example to illuminate the routing procedure. In this example, we suppose that link $C_{11}\leftrightarrow E_{41}$ has failed. Next we look into how node 10.1.1.1 forwards a packet to the destination 10.4.3.2.

- Calculate candidate hops. C_{11} looks up its BRT and obtains the following matched entries:

Destination/Mask	Next Hop
10.4.0.0/255.255.0.0	10.0.1.1
10.4.0.0/255.255.0.0	10.1.3.1
10.4.0.0/255.255.0.0	10.1.4.1

So the candidate hops = {10.0.1.1, 10.1.3.1, 10.1.4.1}.

- Calculate avoiding hops. C11 looks up its NRT and obtains the following matched entries:

Destination/Mask	Next Hop
10.4.0.0/255.255.0.0	10.0.1.1

So the avoiding hops = {10.0.1.1}.

- Calculate applicable hops. The applicable hops are candidate hops minus avoiding hops. So the applicable hops = {10.1.3.1, 10.1.4.1}.
- The priority of the next hop 10.1.3.1 is higher than the next hop 10.1.4.1 according to their path metric. At last, the packet is forwarded to the next hop 10.1.3.1.

D. How to Build Routing Tables

To build a BRT and NRT, FAR needs some knowledge of a network's topology. By leveraging the

regularity in Diamond's topology, switches can build a complete topology view of the entire network through exchanging a little of link information. In FAR, two types of information are exchanged among switches: neighbor relationship and link failures.

FAR builds a BRT based on neighbor relationship, and switches detect neighbor switches by sending and receiving hello messages (heart beat) to and from adjacent switches. FAR builds a NRT based on all the link failures in the entire network. When a switch detects a link failure, the switch broadcasts the link failure to all the other switches. After a period of time, a switch will learn all the link failures in the entire network.

In Diamond, the size of a NRT or BRT is much smaller than that of a routing table in conventional routing method, such as OSPF. The size of a switch's BRT is related to the number of the switch's ports (k). It can be calculated that the size of a core switch's BRT is equal to k and the size of an edge switch's BRT is equal to $9k/4$.

The size of a NRT is related to the number and location of link failures in the network. Suppose that there are n link failures and their distribution is even in the network. We divide link failures into 3 groups by their location: 1) link failures between edge switches and servers; 2) link failures among edge switches; and 3) link failures between edge switches and core switches. Each group has $n/3$ link failures. According to the rules of building NRTs above, we can calculate how many NRT entries a link failure will elicit. A link failure in group 1 only breaks the related server's communication, so it has no effect on any NRT. A link failure in group 2 will elicit at most 2 NRT entries on an edge switch, so group 2 will generate at most $2n/3$ NRT entries on each edge switch. A link failure in group 3 will elicit 1 NRT entry on each edge switch of opposite line and on only 1 edge switch of the same side in each pod, so group 3 will generate about $n/6$ NRT entries on each edge switch. In total, the n link failures will elicit $5n/6$ NRT entries.

If we use conventional routing method in the same Diamond network, such as OSPF, each network segment should have a corresponding route entry in a switch's routing table and there are $k^3/2 + k^2$ segments in a Diamond network. So in OSPF, a switch will have $k^3/2 + k^2$ route entries.

Suppose in a large-scale Diamond network composed of 2,880 48-port switches and 27,648 servers, there are 300 link failures. The size of routing tables in FAR and OSPF is shown in Table. II.

TABLE II: THE SIZE OF ROUTING TABLES IN FAR AND OSPF

Routing Table	Edge Switch	Core Switch
BRT	108	48
NRT	250	0
OSPF Routing Table	57,600	57,600

Table II shows that a FAR routing table is much smaller than an OSPF routing table in a Diamond network, so looking up a FAR routing table is much faster than looking up an OSPF routing table.

V. EVALUATION

In this section, we evaluate Diamond's performance, aggregate throughput and ETE delay, by comparing with Fat-tree through OPNET simulations. OPNET modeler 14.5 is used in simulations.

A. Network Modeling

In the OPNET simulations, Diamond and Fat-tree switches are developed based on the standard layer-3 Ethernet switch model. The routing algorithms for Diamond and Fat-tree are implemented as two process models in the standard layer-3 Ethernet switch model, and the two process models are placed over the ip_encap process model, similar to other routing protocols such as OSPF and ISIS. We modify the standard IP routing model slightly and apply the destination-based multi-path load balancing to our routing algorithm.

2 scenarios are built for the two architectures with different sizes. In scenario 1, a diamond network and a Fat-tree network are built using 20 switches and 16 servers, respectively, as shown in Fig. 3 and Fig. 4.

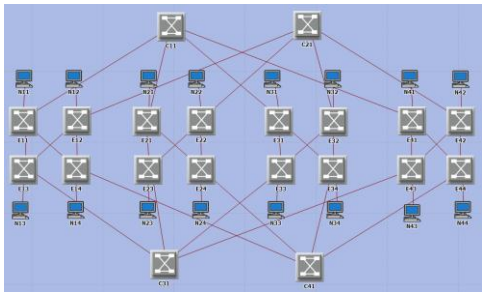


Fig. 3. A diamond simulation network with 4-port switches

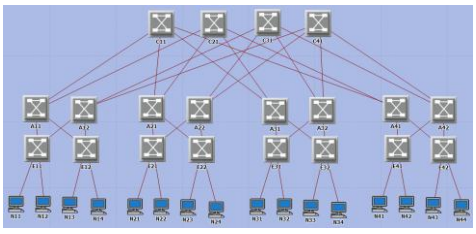
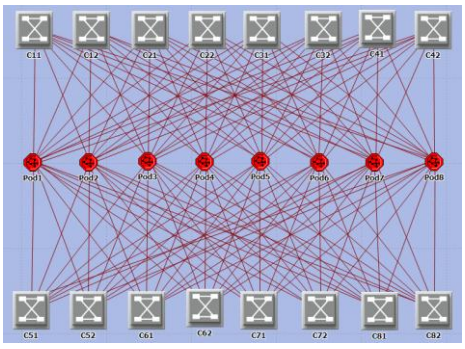
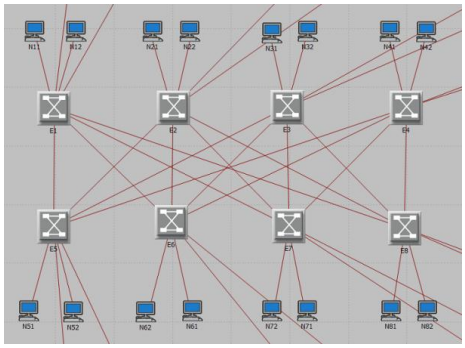


Fig. 4. A Fat-tree simulation network with 4-port Switches



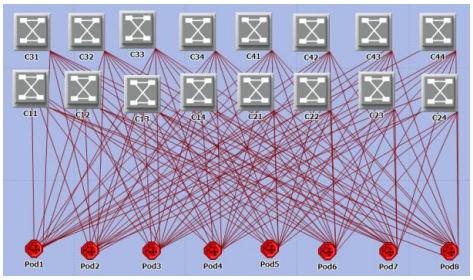
(a) The main view



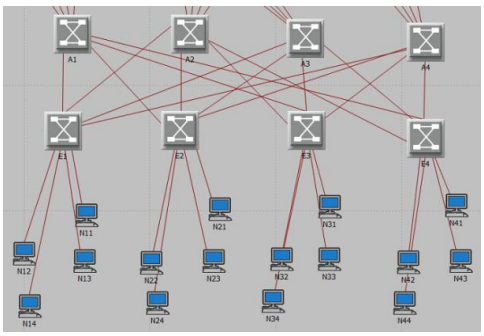
(b) The Pod view

Fig. 5. The diamond simulation network with 8-port switches

In scenario 2, we build a Diamond network and a Fat-tree network using 80 switches and 128 servers, respectively, as shown in Fig. 5 and Fig. 6. Clearly, each pod is put into a logical subnet.



(a) The main view



(b) The Pod view

Fig. 6. The fat-tree simulation network with 8-port switches

B. Aggregate Throughput

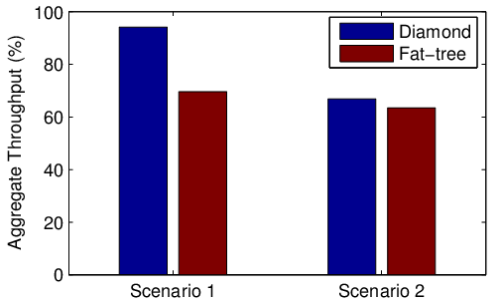


Fig. 7. Aggregate throughput

To measure network aggregate throughput, we pair off all of the servers randomly. Two servers in one pair send packets to each other with a constant speed in simulations. Servers are set to send packets with a rate of 100 Mbits/sec(10,000 packets /sec) in the period of simulation. We measure the average traffic for each server and accumulate them together as the network aggregate throughput. Fig. 7 shows the aggregate throughput as the percentage of the maximum theoretical throughput, i.e., the ideal bisection bandwidth.

Fig. 7 shows that the aggregate throughput of Diamond is larger than that of Fat-tree. With the expansion of network size, the aggregate throughput of Fat-tree is gradually approaching that of Diamond.

C. ETE Delay

In this section, we measure a traffic flow's ETE delay in the Diamond and Fat-tree network of scenario 2. First, we measure the average ETE delay of intra- and inter-pod communication, respectively. Next, we calculate the total average ETE delay in the entire network.

We create a group of traffic flows from Pod1.N11 to the other 15 servers in pod 1 for the two networks in scenario 2, respectively, then measure the average ETE delay of this group as the average ETE delay of intra-pod communication. We create a group of traffic flows from Pod1.N11 to all the servers in Pod 2 for the two networks, respectively, and then measure the average ETE delay of this group as the average ETE delay of inter-pod communication. The speed of each traffic flow is 120 kbps (100 packets/sec). The simulation time is 300 seconds (the flows last from second 165 to second 300). The average ETE delay is shown in Table III.

TABLE III: ETED ELAY COMPARISON (SEC)

Network Type	Intra-pod Delay	Inter-Pod Delay	Total average Delay
Diamond	4.93×10^{-5}	7.88×10^{-5}	7.53×10^{-5}
Fat-tree	5.31×10^{-5}	8.91×10^{-5}	8.49×10^{-5}

Table III shows that the ETE delay in Diamond is obviously lower than that in Fat-tree, which is because the average route path length in Diamond is shorter than that in Fat-tree. Fig. 8 shows the effect of path length on ETE delay. The letter T and D denote Fat-tree and Diamond, respectively. T1 presents the traffic flow in Fat-tree, T2, T3 and T4 present the 3 traffic flows in Diamond. P1 and P2 denote Pod 1 and Pod 2, respectively.

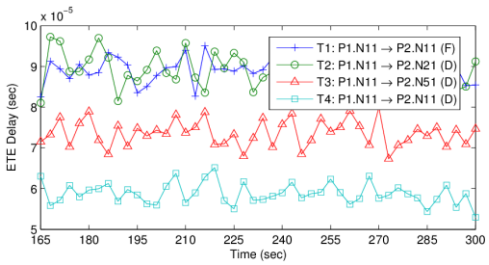


Fig. 8. The relationship between ETE delay and path length

Fig. 8 shows the ETE delay of T1 ≈ T2 > T3 > T4, which is because that the path length of T1 and T2 are 5, and the path length of T3 and T4 is 4 and 3, respectively.

VI. Conclusions

In this paper, we present an improved Fat-tree network named Diamond for large-scale data centers. Simulation and theoretical analysis show that the performance of Diamond exceeds Fat-tree on the sides of route path length and ETE delay. Additionally, a simple and efficient routing method, FAR, is designed for Diamond. In FAR, each switch requires only hundreds of route entries to support a large-scale network with tens of thousands servers, and to build FAR routing tables, switches exchange very few messages.

For future work, we will support virtual machine (VM) migration and multiple tenants in Diamond, because these are the two important features of DCNs.

References

[1] A. Greenberg, J. R. Hamilton, N. Jain, Srikanth Kandula, Changhoon Kim, Parantap Lahiri, *et al*., "Vl2: A scalable and flexible data center network." in *Proc. ACM SIGCOMM'09*, 2009, pp. 51–62.

[2] A. Greenberg, J. Hamilton, D. A. Maltz, and P. Patel, "The cost of a cloud: Research problems in data center networks," in *Proc. ACM SIGCOMM'08*, Aug 2008, pp. 68–73.

[3] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," in *Proc. ACM SIGCOMM'08*, Aug 2008, pp. 63–74.

[4] C. X. Guo, G. H. Lu, D. Li, H. T. Wu, X. Zhang, *et al*., "Bcube: A high performance, server-centric network architecture for modular data centers," in *Proc. ACM SIGCOMM Conference on Data Communication*, Aug 2009,pp. 63–74.

[5] Y. T. Sun, X. L. Song, B. Liu, Q. Liu, and Jing Cheng. (2012). Matrixdcn: A new network fabric for data centers. [Online]. Available: http://tools.ietf.org/html/draft-sun-matrix-dcn-00

[6] R. N. Mysore, A. Pamboris, N. Farrington, N. Huang, *et al.*, "Portland: A scalable fault-tolerant layer 2 data center network fabric," in *Proc. ACM SIGCOMM'09*, Aug 2009, pp. 39–50.

[7] K. Chen, C. X. Guo, H. T. Wu, J. Yuan, *et al.*, "Generic and automatic address configuration for data center networks," in *Proc. ACM SIGCOMM'10*, Aug 2010, pp. 39–50.

[8] N. Farrington, G. Porter, S. Radhakrishnan, H. H. Bazzaz, *et al.*, "Helios: A hybrid electrical/optical switch architecture for modular data centers," in *Proc. ACM SIGCOMM'11*, Aug 2011, pp. 339–350.

[9] G. H. Wang, D. G. Andersen, M. Kaminsky, K. Papagiannaki, T. S. Ng, M. Kozuch, and M. Ryan, "C-through: Part-time optics in data centers," in *Proc. ACM SIGCOMM Computer Communication Review*, Aug 2010, pp. 327–338.

[10] S. Saha, J. S. Deogun, and L. Xu, "Hyscale: A hybrid optical network based scalable, switch-centric architecture for data centers," in *Proc. IEEE International Conference on Communications*, 2012, pp. 2934–2938.

[11] C. X. Guo, H. T. Wu, K. Tan, L. Shi, Y. G. Zhang, and S. W. Lu, "Dcell: A scalable and fault-tolerant network structure for data centers," in *Proc. ACM SIGCOMM Computer Communication Review*, Agu 2008, pp. 75–86.

[12] D. Li, C. X. Guo, H. T. Wu, K. Tan, Y. G. Zhang, and S. W. Lu, "Ficonn: Using backup port for server interconnection in data centers," in *Proc. IEEE INFOCOM'09*, 2009, pp. 2276–2285.

[13] Y. Liao, D.Yin, and L. X. Gao, "Dpillar: Scalable dual-port server interconnection for data center networks," in *Proc. 19th* International *Conference on Computer Communications and Networks*, 2010, pp. 1–6.

[14] D. Guo, T. Chen, D. Li, Y. H. Liu, X. Liu, and G. H. Chen. "BCN: expansible network structures for data centers using hierarchical compound graphs," in *Proc. IEEE INFOCOM'11*, 2011, pp. 61–65.

[15] L ́aszl ́o Gyarmati and T. A. Trinh, "Scafida: A scale-free network inspired data center architecture," in *Proc. ACM SIGCOMM Computer Communication Review*, Aug 2010, pp. 4–12.

[16] A. R. Curtis, T. Carpenter, M. Elsheikh, A. L ́opez-Ortiz, and S Keshav, "REWIRE: An optimization-based framework for unstructured data center network design," in *Proc. IEEE INFOCOM*, 2012, pp. 1116–1124.

[17] J. Hamilton, "An architecture for modular data centers," in *Proc. CIDR'07*, Jan 2007, pp. 306–313.

[18] D. Li, M.W. Xu, H. Zhao, and X. M. Fu, "Building mega data center from heterogeneous containers," in *Proc. 19th IEEE* International *Conference on Network Protocols*, 2011, pp. 256–265.

[19] D. Halperin, S. Kandula, J. Padhye, P. Bahl, and D. Wetherall, "Augmenting data center networks with multi-gigabit wireless links," in *Proc. ACM SIGCOMM'11*, Aug 2011, pp. 38–49.

Yantao Sun received the Ph.D. degree from the Institute of Software Chinese Academy of Sciences in 2006. He is currently a lecturer with the School of Computer and Information Technology, Beijing Jiaotong University. His research interests include cloud computing, datacenter network and network management.

He participated in several national 863 Plan, 973 Plan projects and projects cooperated with Intel, ZTE, Shenghua Group. He published tens of papers on international conferences and journals, wrote 3 national and enterprise standards and 2 textbooks, and applied 4 national invention patents.

Jing Cheng is a CS master student from Beijing Jiaotong University. She received the BS degree in Information Security from Beijing Information Technology University and was awarded the honor of outstanding graduate. Her research interests are data center networks and network simulation.

Qiang Liu received the M.S. and Ph.D. degrees in communication and information systems from the Beijing Institute of Technology in 2004 and 2007, respectively. He is currently a lecturer with the School of Computer and Information Technology, Beijing Jiaotong University.

His research interests lie in mobile communication networks, mobile ad hoc networks and network simulation. He participated in several national defense pre-research projects and international cooperation research projects with Intel during the Tenth Five-Year and the Twelve Five-Year. He published more than 20 papers as the first author. At present, he is leading several research projects.

Weiwei Fang received his B.S. degree in Computer Science from Hefei University of Technology, Hefei, China, in 2003, and Ph.D. degree in Computer Science from Beihang University, Beijing, China, in 2010. From 2010 to 2012, he has been a Post-Doc researcher at School of Computer and Information Technology, Beijing Jiaotong University, Beijing, China. Now he is an assistant professor at Beijing Jiaotong University.

His current research interests include Data Center Networks, Distributed Computing and Wireless Communication. He has coauthored 1 book chapter and more than 30 papers, and served in the Technique Program Committee of more than 30 conferences and workshops.

