FAR: A Fault-avoidance Routing Method for Data Center Networks with Regular Topology

Yantao Sun
School of Computer and Information Technology
Beijing Jiaotong University
Beijing 100044, China
ytsun@bjtu.edu.cn

Min Chen
School of Computer Science & Technology
Huazhong University of Science and Technology
Wuhan 430074, China
minchen@ieee.org

Bin Liu
ZTE Inc.
19 East Huayuan Road
Beijing 100191, China
liu.bin21@zte.com.cn

Shiwen Mao
Department of Electrical and Computer Engineering
Auburn University
Auburn, AL 36849 USA
smao@ieee.org

ABSTRACT

With the widely deployed cloud services, data center networks are evolving toward large-scale and multi-path networks, which cannot be supported by conventional routing methods, such as OSPF and RIP. To alleviate this issue, some new routing methods, such as PortLand and BSR, are proposed for data center networks. However, these routing methods are typically designed for a specific network architecture, and thus lacking adaptability while complex in fault-tolerance. To address this issue, this paper proposes a generic routing method, named fault-avoidance routing (FAR), for data center networks that have regular topologies. FAR simplifies route learning by leveraging the regularity in a topology. FAR also greatly reduces the size of routing tables by introducing a novel negative routing table (NRT) at routers. The operations of FAR is illustrated by an example Fat-tree network and the performance of FAR is analyzed in detail. The advantages of FAR are verified through extensive OPNET simulations.

Keywords

Data Center Network; Routing Method; Regular Topology; Fault-Avoidance Routing; Negative Routing Table

1. 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.

In recent years, some new network architectures have been proposed for data centers [1][2]. These network architectures can support more than tens of thousands of servers with very high internal bandwidth, and some of them can be used to construct non-blocking networks, such as Fat-tree [3], BCube [4] and MatrixDCN [5]. To connect large numbers of servers, many switches or servers with routing functionality are used in a DCN. For example, to accommodate 27,648 servers, a Fat-tree network requires 2,880 switches, each of which has 48 switching ports. It is unlikely that generic routing protocols such as OSPF and IS-IS can be scaled to support several thousands of routers [6], so some specific routing protocols are proposed for these architectures. These specific routing protocols have better performance than the generic routing protocols, because they are delicately designed according to the topological feature of the network architecture.

In the future, an extra-large-scale DCN may be composed of several heterogeneous networks, and each network employs a different architecture [7]. Several network architectures coexist in a DC. From the cost-effective point of view, a desirable routing device should have agility for supporting various mainstream network architectures and the related routing protocols, which enables the reuse of the routing device when the architecture is changed. But in fact, the proposed routing algorithms for various architectures are so different, which causes difficulty to renovate a routing device to support multiple existing routing protocols. Furthermore, these routing methods have poor compatibility with the current routing protocols, since the structure and query method of their routing table, fault-tolerance and routing mechanism are all different with current routing protocols. It's a challenging issue to implement these specific routing protocols through modifying the current routing protocols. In addition, to deal with link failures, these specific routing methods of new network architectures introduce complicated fault-tolerance mechanisms, which makes these routing methods more complex [3] [4].

In this paper, to address the problems above, we propose a generic routing method, named fault-avoidance routing (FAR) method, for DCN. Not limited by a specific type of network architecture, this routing method can be used in any network architectures with regular topologies.

FAR consists of three components, i.e., link state learning unit, routing table building unit and routing table querying unit. In the link state learning unit, FAR exchanges link

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

978-1-4799-1640-5/13/$31.00 ©2013 IEEE

failures among routers to establish a consistent knowledge of the entire network. In this stage, the regularity in topology is exploited to infer failed links and routers. In the routing table building unit, FAR builds up two routing tables, i.e., a basic routing table (BRT) and a negative routing table (NRT), for each router according to the network topology and link states. In the last component, routers forward incoming packets by looking up the two routing tables. The matched entries in BRT minus the matched entries in NRT are the final route entries to be used for an incoming packet.

FAR simplifies the computing of routing tables by leveraging the regularity in topology and decreases the size of routing tables by introducing the NRT at routers. As opposed to the existing routing methods, FAR has the following advantages:

- Each router is only configured one IP address, which greatly simplifies the configuration of a data center network.
- By leveraging the regularity in topology, FAR avoids the problem of network convergence and reduces the complexity of calculating routes. The time of calculating routes is shortened to hundreds of milliseconds from tens of seconds for typical DCN scenarios.
- By introducing NRT, FAR decreases the size of routing tables. In a DC that contains tens of thousands of severs, FAR routing tables only have tens of route entries.
- FAR has good adaptability. It can be used in many kinds of network topologies after slight modifications.

The main contributions of this paper are three-fold: 1) propose a generic routing framework for the DCN with a regular topology; 2) introduce the negative routing table to routing methods, which can reduce the size of routing tables; 3) give a performance analysis and a thorough simulation study of FAR with OPNET.

The remainder of this paper is organized as follows. Section 2 introduces the related research on routing for DCN. Section 3 presents the framework of FAR routing and uses a Fat-tree network to illustrate the operations of FAR. Section 4 analyzes the performance of FAR. Section 5 verifies FAR routing through OPNET simulations. Section 6 concludes for this paper.

2. RELATED WORK

RIP and OSPF are the two generic routing protocols that are widely applied for interior (intra-domain) routing. As a distance-vector based algorithm, RIP works fine only for small-size network, as it uses a hop count of 15 to denote infinity and has the slow convergence or count-to-infinity problem, which makes it unsuitable for large networks. OSPF addresses all RIP shortcomings and thus is better suited for modern large, dynamic networks. But in really large configurations, the huge number of router updates–that flow between routers–can become an issue. In very large OSPF networks, topology convergence can be delayed, while routers exchange link-state messages, update databases, and recalculate routes. To address this issue, an OSPF network is divided into many areas and each area calculate routes independently. But in practice, it is very difficult to divide a data center network, such as a Fat-tree network, into multiple OSPF areas, so OSPF is not suitable to a large-scale data center network.

To support large data center networks, some layer-2 routing methods were proposed. In these layer-2 switching methods, such as TRILL[8] and Cisco's FarbricPath[9], some layer-3 routing technologies are applied for Ethernet frames' forwarding in layer-2 network. TRILL is an IETF standard that is used in devices called RBridges (routing bridges) or in TRILL Switches, which provide multi-path forwarding for Ethernet frames. In TRILL, RBridges compute the shortest path and equal-cost paths in layer 2 by using TRILL IS-IS, which is a link-state routing protocol similar to IS-IS routing protocol. MAC-in-MAC encapsulated packets are forwarded to the destination host via the switched network comprising RBridges. FabricPath is a similar but private technology provided by Cisco.

Another layer-2 routing solution is SEATTLE, which uses a link-state routing protocol to establish a routing path between switches [10]. Unlike TRILL, SEATTLE uses the global switch-level view provided by a link-state routing protocol to form a one-hop DHT. The DHT stores IP to MAC mapping and MAC to host location mapping of each host in switches. SEATTLE converts an ARP request to a unicast-based message to obtain a destination host's MAC address and destination switch that the destination host is connected to, and then forwards packets to the destination switch. At last the destination switch forwards packets to the destination host.

SPAIN [11] and NetLord [12] demonstrate a new thinking about layer-2 interconnection based on existing switch devices in an arbitrary topology. Within these methods, a set of paths is pre-computed offline for each pair of source-destination hosts by exploiting the redundancy in a given network topology. Then, these paths are merged into a set of trees, and each tree is mapped onto a separate VLAN. In this way, a proxy application is installed on the hosts, and the proxy chooses several VLAN paths transmitting packets to the destination host. The advantage of this method is that multipath is implemented, and routing load is balanced on multiple paths in an arbitrary topology. Its drawbacks are inflexibility to changes in topology and the modifications to hosts.

Besides these generic routing algorithms, researchers proposed some specialized routing algorithms for some special network architectures. These routing algorithms leverage the regularity in the topology, so they have higher efficiency on route computing and packet forwarding.

BCube[2] is a sever-centric network architecture in which servers are responsible for both computing service and routing function. In BCube, a source routing protocol called BSR is deployed on servers. BSR has the abilities of load balance and fault-tolerance. When a new flow comes, the source sends probe packets over multiple parallel paths and selects the best path according to probe responses. In general, using a source routing protocol in a large-scale network may result in too much network overhead and too long connection time.

Fat-tree [3] uses two-level routing tables to spread outgoing traffic on multiple equal cost paths. When making a routing decision, a switch looks up a main routing table firstly. If no route is hit, then the switch looks up a small secondary table. To implement fault-tolerance in rout-

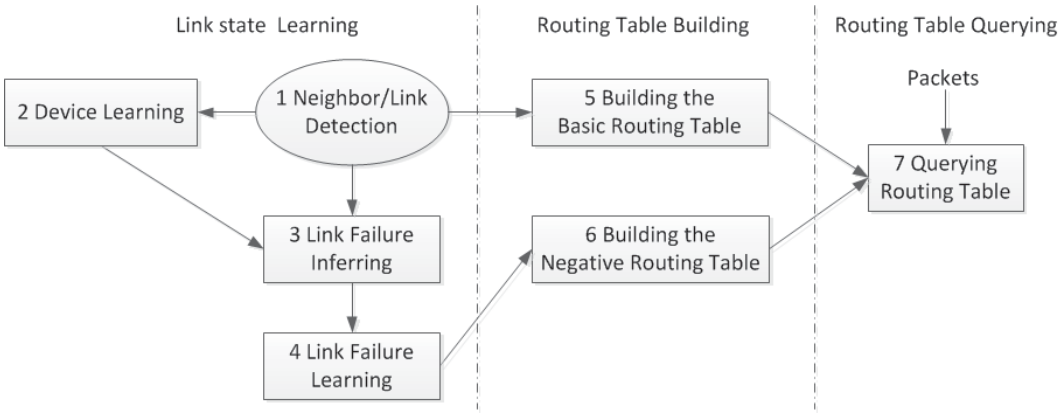

Figure 1: The Routing Framework of FAR

ing procedure, Fat-tree detects link failures by a Bidirectional Forwarding Detection session (BFD) [13] and broadcasts those failures between switches. When assigning new flows, switches check whether the intended output port corresponds to one of link failures and try to avoid link failures if possible.

PortLand is a layer-2 network solution specifically for the Fat-tree network [14]. It uses a lightweight location discovery protocol (LDP) that allows switches to discover their location in the topology. In PortLand, every end host is assigned an internal pseudo MAC (PMAC) that encodes the location of the end host. Different with other layer-2 technologies, PortLand leverages the knowledge of network structure within layer-2 routing, which helps that PortLand has smaller switching overhead, more efficient forwarding, and higher fault tolerance than others. Its drawback is that PortLand is specified for the Fat-tree network only.

Compared with generic routing methods, topology-aware routing algorithms leverage the regularities in topology in route computing and thus have a higher efficiency. The shortages of those topology-aware routing algorithms are: 1) they are closely designed for the special topology of networks; 2) their fault-tolerance mechanisms are complicated.

3. FAR ROUTING

We propose a generic distributed routing method, fault-avoidance routing (FAR), specifically designed for a data center network with regular topology. FAR simplifies route computing by leveraging the regularity in network topologies and solves the problem of lack of adaptability in existing routing methods of data center networks. Compared with other routing methods, FAR is simpler and more efficient, and has good fault-tolerance performance.

FAR requires that a data center network has a regular topology, and network devices, including routers, switches, and servers, are assigned IP addresses according to their location in the network. In other words, we can locate a device in the network according to its IP address. Note that these assumptions are generally true for most data center networks. To run FAR in a data center network, each router or switch with routing functionality should be deployed with a FAR instance.

FAR is divided into three parts: 1) the link state learning unit, which learns all the link failures in the entire network; 2) the routing table building unit, which builds up routing tables according to learnt link states; and 3) the routing table querying unit, which looks up routing tables to forward packets. The framework of FAR is shown in Fig.1.

The link state learning unit consists of 4 modules. In the neighbor/link detection module (M1), switches detect their neighbor switches and connected links by a heartbeat mechanism. The device learning module (M2) learns all the active switches in the entire network, and then infers failed switches according to the network topology. In the link failure inferring module (M3), switches infer their invisible neighbors and related failed links. If a link between router S_a and its neighbor S_b breaks, then S_b can't be detected by S_a through a heartbeat mechanism. In this case, S_b is called an invisible neighbor of S_a. In the link failure learning module (M4), every switch learns all the failed links in the entire network.

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 the routing table in conventional routing methods, telling FAR what paths are available to reach a destination node. Oppositely, an NRT tells what paths FAR should avoid because those paths pass through some failed links. FAR looks up the two routing tables to get the final applicable paths. BRT building module (M5) builds up a BRT for a router depending on its neighbors and links detected in M1. NRT building module (M6) builds up an NRT for a router depending on the failed switches and links learned in M2 and M4. Routing table querying module (M7) looks up both an NRT and a BRT, and then combines query results together to forward incoming packets.

3.1 Example Fat-tree Network

The example Fat-tree network is built with 4-port switches, as shown in Fig.2. The Fat-tree network is composed of layer-3 switches. There are 4 layers in the Fat-tree network. The top layer is the core layer, and the other layers are the aggregation layer, edge layer and server layer, respectively. Except for core switches, all the other devices are partitioned into many pods.

In the example network, devices are elaborately assigned IP addresses according to their locations. Aggregation switches, edge switches and servers are given addresses in the form of 10.*pod*.0.*switch*, 10.*pod.switch*.1 and 10.*pod.switch.dev*,

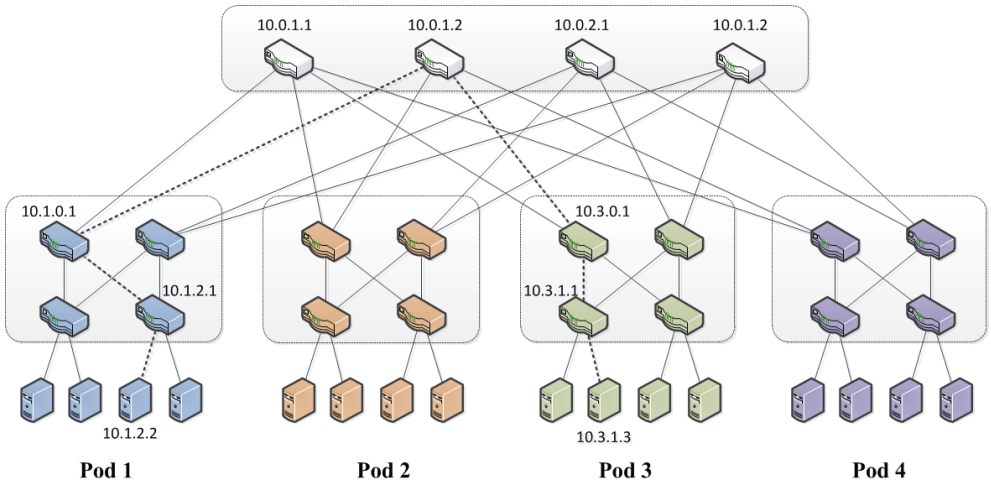

Figure 2: The example Fat-tree Network

respectively, where *pod* denotes the pod number, *switch* denotes the switch's position in the pod and *dev* denotes a server's position in the pod. Core switches are given addresses in the form of $10.0.i.j$.

In the following paragraphs, we will take the Fat-tree network as an example to introduce the principles of FAR.

3.2 Learn Link States

To avoid failed links when forwarding packets, every switch needs to know the states of all links, active or failed. FAR spreads all the switches and link-state changes to each switch by a flooding method. To avoid a message looping in the network, each switch only forwards unknown switches and link failures to their neighbors. The flooding method insures that each switch can learn all the switches and link failures since a switch will receive a message multiple times from its neighbors. The procedure of learning link states is divided into four steps.

1) Neighbor Discovery

Adjacent switches send hello messages (heartbeat) to each other for detecting their neighborhood relationship. M1 sends hello messages periodically to all of its ports and receives hello messages from its neighbor routers. M1 detects neighbor switches and directly-connected links according to received hello Messages and stores these neighbors and links in a local database.

2) Device Learning

FAR learns all the switches in the entire network through device announcement (DA) messages and device link request (DLR) messages. When a switch starts, it sends a DA message announcing itself to its neighbors and a DLR message requesting the knowledge of switches and links from its neighbors. When M2 receives a DA message, it stores unknown switches encapsulated in the message into its local database and then forwards them to its neighbors except for the incoming one. M2 replies a DLR message with a DA message that encapsulates all the routers in its database. Device announcements will be repeatedly published by a long period such as 30 minutes.

When M2 has learned all the active switches, it infers failed switches in the network by leveraging the regularity in topology. For example, we suppose that the node 10.1.0.1 fails, so it cannot be detected by node 10.1.1.1 through the heartbeat mechanism. But according to the regularity in topology and assignment rules of IP address in the Fat-tree network, 10.1.1.1 can infer that there exists a failed aggregation switch whose IP address is 10.1.0.1 and the aggregation switch is in failure.

3) Link Failure Inferring

Because a device's location has been coded into its IP address, it can be determined whether two routers are adjacent according to their IP addresses. For example, we can infer that the core switch 10.0.1.1 (S_a) and the aggregation switch 10.1.0.1 (S_b) are adjacent. If S_a learns the existence of S_b through a DA message but S_a can't detect S_b through M1 module, S_a can infer that S_b is an invisible neighbor of S_a and a failed link lies between S_a and S_b. Based on this idea, M3 infers all the invisible neighbors of a router and the related link failures.

4) Link Failure Learning

FAR requires the knowledge of link failures in the entire network. It learns link failures through link failure announcement (LFA) messages and DLR messages. When a switch starts, it sends a DLR message to request the knowledge of routers and links from its neighbors. M4 will answer an LFA message that encapsulates all the link failures in its database. If a link fails or recovers, M4 will also broadcast the change to its neighbors through an LFA message. When M4 receives an LFA message, it updates its local link-state database and then forwards them to its neighbors except for the incoming one.

3.3 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. For example, the core switch 10.0.1.1(S_a) knows that its port 1 is connected to the aggregation 10.1.0.1(S_b) and the subnet 10.1.0.0/16. When S_a receives a packet whose destination is $10.1.*.*$, it will forward the packet to S_b from its port 1. In order to implement such a routing, S_a will create a route entry whose destination is 10.1.0.0/16, whose next hop is S_b and whose corresponding interface is port 1 in its BRT. Since IP addresses of network devices are continuous,

FAR only creates one route entry for a group of destination addresses that have the same network prefix by means of route aggregation technology.

Module M5 builds up a BRT for a router depending on the detected neighbors and links in M1. Every switch in a Fat-tree network clearly knows how it forwards a packet, so it's easy to build BRTs for the switches. The forwarding policy of a switch is described as follows: When a packet arrives at an edge switch, if the destination of the packet lies in the subnet that the switch connects with, then it directly forwards the packet to the destination server through layer-2 switching; otherwise, the switch forwards the packet to any of the aggregation switches in the same pod. When a packet arrives at an aggregation switch, if the destination of the packet lies in the current pod, then the switch forwards the packet to the corresponding edge switch that the destination server connects to; otherwise, the switch forwards the packet to any of core switches that it connects to. If a core switch receives a packet, it forwards the packet to the corresponding aggregation switch that lies in the destination pod. The forwarding policy discussed above is easily expressed through a BRT. For example, the BRT of the aggregation switch 10.1.0.1 is as follows:

Destination/Mask	Next Hop
10.1.1.0/255.255.255.0	10.1.1.1
10.1.2.0/255.255.255.0	10.1.2.1
10.0.0.0/255.0.0.0	10.0.1.1
10.0.0.0/255.0.0.0	10.0.1.2

The BRT of an edge or core switch is similar to aggregation switches, which are omitted for brevity.

The number of route entries in a BRT of a switch is equal to the number of its neighbor switches, which is from several to tens, so the size of a BRT is very small and looking up an entry in such a small table will be very fast.

3.4 Negative Routing Table

Because a BRT doesn't consider link failures, the routing paths calculated by a BRT can't avoid failed links. To solve this problem, one method is to tell a switch what hops, i.e. paths, it can forward packets to, and another method is to tell a switch what hops it can't. Conventional routing methods, such as OSPF and RIP, are all based on the first method. In FAR, the second method is used. FAR uses a NRT to exclude the routing paths that pass through some failed links from the paths calculated by a BRT. The M6 module builds up a NRT for a router depending on the learned device and link failures in M2 and M4.

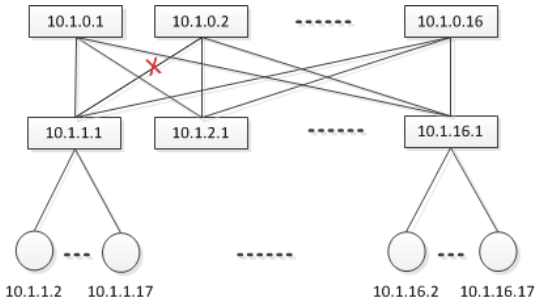

Figure 3: An example of multiple paths network

Compared with conventional routing method, using an NRT to avoid failed links in a multiple paths network is simple and can decrease the size of routing tables remarkably. For example, in Fig. 3, if the network has no failures, the routing table of node 10.1.16.1 has 16 route entries as follows:

Destination/Mask	Next Hop
10.1.0.0/255.255.0.0	10.1.0.1
10.1.0.0/255.255.0.0	10.1.0.2
...	
10.1.0.0/255.255.0.0	10.1.0.16

If the link between node 10.1.1.1 and 10.1.0.2 fails, in order to avoid this failed link for communications from the subnet 10.1.16.0/24 to the subnet 10.1.1.0/24, in conventional routing methods the node 10.1.16.1 should add 15 additional route entries as follows:

Destination/Mask	Next Hop
10.1.1.0/255.255.255.0	10.1.0.1
10.1.1.0/255.255.255.0	10.1.0.3
...	
10.1.1.0/255.255.255.0	10.1.0.16

The above entries tell the switch that the traffic to subnet 10.1.1.0/24 can be forwarded to 10.1.0.1, 10.1.0.3, ..., and 10.1.0.16 except for 10.1.0.2.

But in FAR, only one route entries is needed:

Destination/Mask	Next Hop
10.1.1.0/255.255.255.0	10.1.0.2

It tells the router, for the traffic to subnet 10.1.1.0/24, the next hop cannot be 10.1.0.2.

Module M6 builds up an NRT for a switch based on link and device failures of the entire network. If a link or device comes back up, related route entries should be removed from NRTs. In a Fat-tree network, only aggregation switches and edge switches require NRTs. Next we illustrate how a switch builds its NRT in a Fat-tree network. As a device failure can be treated as a group of link failures, we don't discuss device failures in this paper. We also don't discuss recovering of links or devices since it just is a reverse procedure of link failures.

3.4.1 Single Link Failure

In a Fat-tree network, links can be classified into three types according to their locations: (1) servers to edge switches; (2) edge to aggregation switches; (3) aggregation to core switches. Link failures between servers to edge switches only affect the communication of the corresponding servers and don't affect any switch's routing tables. So we only need to consider the failures of the second and third type of links.

1) Edge-to-aggregation link Failures

If a link between an edge switch, such as 10.1.2.1(S_a), and an aggregation switch, such as 10.1.0.1(S_b), fails, it may affect three types of communications:

- A communication whose source lies in the same subnet with S_a, but whose destination does not. In this case, the link failure will only affect the routing of S_a. As this link is attached to S_a directly, S_a only needs to delete the route entries whose next hops are S_b in its BRT and doesn't add entries to its NRT.

- A communication that its source and destination both lie in the same pod with S_a, and the destination lies in the same subnet with S_a but the source does not. In this case, the link failure will affect the routing of all the edge switches in the pod except for S_a. When an edge switch, such as 10.1.1.1, learns the link failure, it will add a route entry into its NRT:

Destination/Mask	Next Hop
10.1.2.0/255.255.255.0	10.1.0.1

- A communication whose destination lies in the same subnet with S_a and whose source lies in another pod. In this case, the link failure will affect the routing of all the edge switches except for those lie in the same pod with S_a. When an edge switch, such as 10.3.1.1, learns the link failure, because all the routes that pass through 10.3.0.1 to S_a will certainly pass through the link between S_a and S_b, 10.3.1.1 needs to add a route entry to its NRT in order to avoid 10.3.0.1:

Destination/Mask	Next Hop
10.1.2.0/255.255.255.0	10.3.0.1

2) Aggregation-to-core link failures

If a link between an aggregation switch, such as 10.1.0.1(S_a), and a core switch, such as 10.0.1.2(S_b), fails, it may affect 2 types of communications:

- A communication whose source lies in the same pod with S_a and whose destination lies in another pod. In this case, only S_a's routing will be affected. As the failed link is attached to S_a directly, S_a only needs to delete the route entries whose next hops are S_b in its BRT.

- A communication whose destination lies in the same pod with S_a and whose source lies in another pod. In this case, the link failure will affect the routing of all the aggregation switches except for those lie in the same pod with S_a. When an aggregation switch, such as 10.3.0.1, learns the link failure, it needs to add a route entry to its NRT:

Destination/Mask	Next Hop
10.1.0.0/255.255.0.0	10.0.1.2

3.4.2 Combination of Link Failures

We call two aggregation switches at the same position in different pods as a pair of inter-pod aggregation switches. For example, 10.1.0.1 and 10.2.0.1 are a pair of inter-pod aggregation switches. There exist multiple paths between a pair of inter-pod aggregation switches. If all the paths between a pair of inter-pod aggregation switches are broken, then the two aggregation switches cannot play as intermediate nodes for the communications between the two pods. In this case, the combination of a group of link failures will affect NRTs of some edge switches. In a Fat-tree network, only combinations of aggregation-to-core link failures affect NRTs, and affect only NRTs of edge switches.

We take the pair of 10.1.0.1(S_a) and 10.2.0.1(S_b) as an example. C_a and C_b denote the set of core switches that the failed uplinks of S_a and S_b connect to, respectively. C_i denotes the intermediate core switches between S_a and S_b. If $C_a + C_b = C_i$, then all the paths between S_a and S_b via core switches are broken. In this case, we need to add a route entry to the NRT of each edge switches in pod 1:

Destination/Mask	Next Hop
10.2.0.0/255.255.0.0	10.1.0.1

And we need to add a route entry to the NRT of each edge switches in pod 2:

Destination/Mask	Next Hop
10.1.0.0/255.255.0.0	10.2.0.1

When an edge switch receives an aggregation-to-core link failure, the edge switch calculates its NRT according to not only the single link failure but also combinations with other link failures. Because the related aggregation switch of the failed link can compose multiple pairs with other aggregation switches in different pods, the edge switch will calculate its NRT for each combination of link failures corresponding to each pair of aggregation switches.

3.5 Lookup Routing Tables

FAR looks up both the BRT and the NRT to decide the next hop for a forwarding packet. Firstly, 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. Secondly, FAR looks up route entries in an NRT also taking the destination address of the forwarding packet as criteria. In this lookup, there is no regard to longest prefix match, and any entry that matches the criteria would be selected and composed of a set of avoiding entries. Thirdly, the candidate entries minus the avoiding entries represent the set of applicable entries. At last, FAR sends the forwarding packet to any one of the applicable entries. If the set of applicable entries is empty, the forwarding packet will be dropped.

We take the following example to illuminate the routing decision procedure. In this example, we suppose that the link between 10.0.1.2 and 10.3.0.1 has failed. Next we look into how node 10.1.0.1 forwards a packet to the destination 10.3.1.3.

1) Calculate candidate hops. 10.1.0.1 looks up its BRT and obtains the following matched entries:

Destination/Mask	Next Hop
10.3.0.0/255.255.0.0	10.0.1.1
10.3.0.0/255.255.0.0	10.0.1.2

So the candidate hops = $\{10.0.1.1, 10.0.1.2\}$.

2) Calculate avoiding hops. 10.1.0.1 looks up its NRT and obtains the following matched entries:

Destination/Mask	Next Hop
10.3.0.0/255.255.0.0	10.0.1.2

So the avoiding hops = $\{10.0.1.2\}$.

3) Calculate applicable hops. The applicable hops are candidate hops minus avoiding hops. So the applicable hops = $\{10.0.1.1\}$.

4) Finally, the packet is forwarded to the next hop 10.0.1.1.

Table 1: Required messages in a Fat-tree network

Message Type	Scope	Size	Rate	Bandwidth
Hello	Between adjacent switches	< 48 bytes	10 messages/sec	<4 kbps
DLR	Between adjacent switches	< 48 bytes	Produce one when a router starts	48 bytes
DA	In the entire network	< 48 bytes	The number of switches (2,880) in a period	1.106M
LFA	In the entire network	< 48 bytes	Produce one when a link fails or recovers	48 bytes

4. EVALUATION

In this section we evaluate FAR's performance with respect to the number of control messages, the route calculating time and the size of routing tables. A Fat-tree network composed of 2,880 48-port switches and 27,648 servers is used for the test in this section.

4.1 The number of control messages required by FAR

FAR exchanges a few messages between routers and only consumes a little network bandwidth. Tab. 1 shows the required messages in the test Fat-tree network.

The last column of Tab. 1 presents the bandwidth consumed by one type of message on each link. From Tab. 1, we can see that hello messages are only exchanged among neighbor switches, so hello messages use less than 4kbps bandwidth. An LFA message is produced when a link fails or recovers from failure and a DLR message is produced only when a switch is booted up, so the two types of messages are very few and consume very little bandwidth. Each switch will produce a DA message in a DA period and the message will pass through a link no more than one time when the message is spread throughout the entire network, so the number of DA message passed through a link is no more than the number of switches in a DA period. The maximum bandwidth consumed by DA messages in a DA period is no more than $the\ number\ of\ switches \times the\ size\ of\ DA\ message = 2,880 \times 48 \times 8bits = 1.106Mbits$. Because a DA period is very long (30 minutes), DA messages use little bandwidth.

It can be concluded that even in a very large data center with about 30,000 servers and 2,880 switches, FAR produces a few number of messages and uses little bandwidth.

4.2 The calculating time of routing tables

A BRT is calculated according to the states of its neighbor routers and attached links. An NRT is calculated according to device and link failures in the entire network. So FAR does not calculate network topology and has no problem of network convergence, which greatly reduces the calculating time of routing tables. In FAR, a router can calculate its BRT and NRT in several milliseconds. But in OSPF, the calculating time may require several minutes for a large network. Dijkstra algorithm is used in OSPF to calculate the shortest path tree. Using Dijkstra algorithm, it spent 15 seconds to calculate the shortest path tree for the test Fat-tree network in our experiment running on a machine with an Intel I7 2.8GHz Dual-core CPU, 8G memory and Windows 7 OS.

The detection and spread time of link failures is very short in FAR. Detection time is up to the interval of sending Hello message. In FAR, the interval is set to 100ms, and a link failure will be detected in 200ms. The spread time between any pair of routers is less than 200ms.

If a link fails in a data center network, FAR can detect it, spread it to all the routers, and calculate routing tables in no more than 500ms.

4.3 The size of routing tables

For the test Fat-tree network, the sizes of BRTs and NRTs are shown in Tab. 2.

Table 2: The size of routing tables in FAR

Routing Table	Core Switch	Aggregation Switch	Edge Switch
BRT	48	48	24
NRT	0	14	333

The BRT's size at a switch is determined by the number of its neighbor switches. For example, in the test network, a core switch has 48 neighbor switches (aggregation switch), so it has 48 entries in its BRT.

Only aggregation and edge switches have NRTs. The NRT size at a switch is related to the number of link failures in the network. Suppose that there are 1000 link failures in the test Fat-tree network. The number of failed links is 1.2% of total links, which is a very high failure ratio. We suppose that link failures are uniformly distributed in the entire network. Next, we analyze NRTs' size of aggregation and edge switches.

The link failures that affect NRTs of edge switches are mainly the link failures between edge and aggregation switches, and the number of these link failures is about 1000/3 = 333, due to the uniform distribution assumption. Because each link failure of this type will generate an NRT entry in an edge switch, the NRT size at an edge switch is about 333. The link failures that affect NRTs of aggregation switches are mainly the link failures between aggregation and core switches, and the number of these link failures is about 1000/3 = 333 too. We put aggregation switches into groups by their locations or IP addresses. All the aggregation switches whose IP addresses have the same forth number are put into a group, so aggregation switches in the test network are divided into $N/2 = 24$ groups and each group has about 333/24 = 14 link failures. An uplink failure of an aggregation switch will elicit one NRT entry on other aggregation switches in the same group, so the NRT size of an aggregation switch is about 14.

Next we measure the routing table of a Fat-tree network running OSPF in an OPNET simulation. The version of

OPNET is 14.5. This Fat-tree network is composed of 20 4-port switches and 16 servers, as shown in Fig. 4. To simplify network provision, we replace layer-3 switches with routers in the core and aggregation layers, and replace a layer-3 edge switch with a router plus a layer-2 switch. We assign IP addresses to servers and each interface of routers by the auto-assign IP addresses mechanism of OPNET.

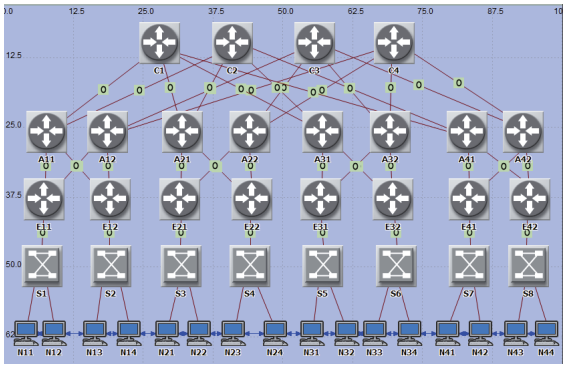

Figure 4: A Fat-tree network running OSPF

We configure OSPF as the routing protocol for the Fat-tree network and then execute it to compute routing tables. When the simulation is over, each router has 40 valid entries in its routing table and each entry corresponds to a network segment. In this network, there are 40 network segments totally and each interface of a router is attached to a network segment. If we run OSPF in the test Fat-tree network, there will be 56,448 network segments, so each router would has 56,448 entries in its routing table, as shown in Tab. 3.

Table 3: The size of routing table in OSPF

The Scale of Network	Core Switch	Aggregation Switch	Edge Switch
A Fat-tree network with 4-port switches	40	40	40
A Fat-tree network with 48-port switches	56,448	56,448	56,448

Comparing Tab. 2 and Tab. 3, we can conclude that the size of routing tables in FAR is much smaller than that in conventional routing mothods, such as OSPF.

5. VERIFYING FAR WITH SIMULATIONS

We verify FAR with OPNET simulation in this section. The version of OPNET is 14.5. The FAR switches are developed based on the standard layer-3 Ethernet switch model. FAR is implemented as a process model in the standard layer-3 Ethernet switch model and the process model is placed over the ip_encap process model, similar to other routing protocols such as OSPF and ISIS. We modified the standard IP routing model slightly and applied the destination-based multi-path load balancing to our routing algorithm.

Fig. 5 is the state transition diagram of the FAR process. At the entrance of Neighbor Detection (ND) state, ND_TIMEOUT and SELF_INTR_HELLO self-interruptions are created. At the exit of ND state, the DA message and DLR message are broadcast to its neighbor switches. When

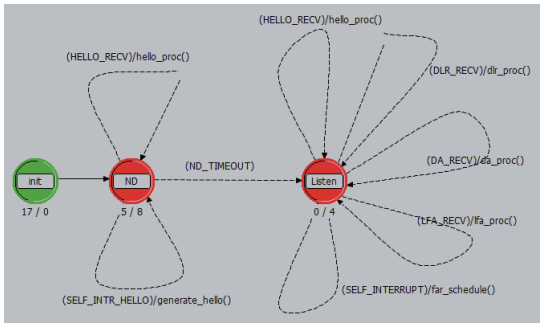

Figure 5: State transition of the FAR Process

there is an ND_TIMEOUT interruption, FAR process goes into the Listen state. The far_schedule procedure is responsible for detecting link and node failures and for generating Hello, DA and LFA messages periodically.

In the simulation, we build up a Fat-tree network using 4-port FAR switches, as shown in Fig. 6. In the figure, red solid lines between network nodes represent 100Mbps links. The blue thin dotted line presents a traffic flow form node N13 to node N32. The blue and red thick dotted lines present two routes for the traffic flow.

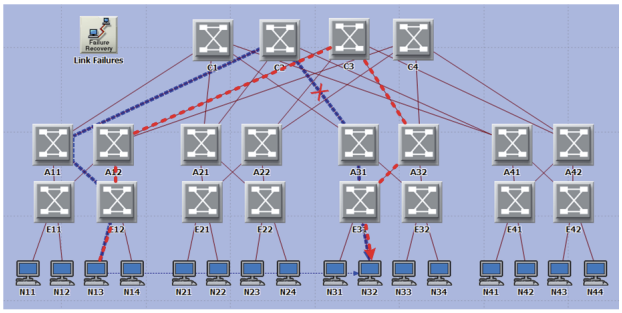

Figure 6: A Fat-tree network running FAR

The traffic starts at second 165 and finishes at second 250, as shown in Fig. 7. At first, the traffic is forwarded along the blue thick dotted line. At second 200, the link between C2 and A31 breaks, then FAR recalculates routing tables and the traffic is switched to the red thick dot line. Because the traffic is interrupted about 200ms when the route is switched at second 200, the number of received packets in second 200 drops down and then recovers to the normal rate immediately.

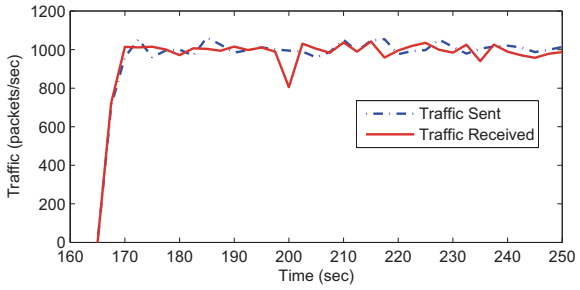

Figure 7: Traffic Sent & Received in FAR

Through the simulation, we demonstrate that FAR works well in a Fat-tree network. It can forward packets properly and respond to link failures quickly by avoiding the failed link in several hundred milliseconds.

6. CONCLUSIONS

In this paper, we proposed a generic routing method, FAR, for data center networks. FAR leverages the regularity in network topology to effectively simplify the calculating procedure of routes, to decrease the size of routing tables, and to improve the efficiency of routing. Different from conventional routing methods, FAR uses two routing tables in its routing procedure, a BRT and a NRT. When making a routing decision, FAR firstly looks up the BRT and obtains a set of candidate route entries, then looks up the NRT and obtains a set of avoiding route entries. The final route entries are obtained by removing the avoiding route entries from the candidate route entries. Compared with alternative routing methods, FAR has many obvious advantages:

1) The network convergence and calculating the shortest path tree require a long time in most routing methods, but FAR doesn't have these requirements, which effectively shortens the time of calculating routes, accelerates its response time to network changes, and relieves the computing burdens of a router.

2) In FAR, the calculating of the BRT and NRT is very simple and requires only a few computations. So it can be quickly completed in several milliseconds, even for very large scale data center networks.

3) The size of routing tables in FAR is very small. A BRT only has tens of entries and an NRT has no more than hundreds of entries. It is very fast to look up routing tables in FAR.

4) FAR has very good adaptability. It can be used in many kinds of data center network topologies with slight modifications.

For future work, we will focus on extending FAR to support virtual machine (VM) migration and multiple tenants, since these are the two important features of data center networks.

7. ACKNOWLEDGMENTS

This work is supported by the State Key Laboratory of Rail Traffic Control and Safety, Beijing Jiaotong University (Contract No. RCS2012ZT007), the ZTE-BJTU Collaborative Research Program (No. K11L00190) and the Chinese Fundamental Research Funds for the Central Universities (No. K12JB00060). Shiwen Mao's research is supported in part by the US National Science Foundation under Grants CNS-0953513, CNS-1320664, and IIP-1266036.

8. REFERENCES

[1] M. Chen, Y. Wen, H. Jin, and V. Leung. Enabling technologies for future data center networking: A primer. *IEEE Network, Special Issue on Cloud and Data Center Performance*, July 2013.

[2] Yantao Sun, Jing Chen, Konggui Shi, and Qiang Liu. Data center network architecture. *ZTE Communications*, 11(1):54–61, Mar 2013.

[3] Mohammad Al-Fares, Alexander Loukissas, and Amin Vahdat. A scalable, commodity data center network architecture. In *ACM SIGCOMM'08*, pages 63–74. ACM, Aug 2008.

[4] Chuanxiong Guo, Guohan Lu, Dan Li, Haitao Wu, Xuan Zhang, Yunfeng Shi, Chen Tian, Yongguang Zhang, and Songwu Lu. Bcube: a high performance, server-centric network architecture for modular data centers. In *ACM SIGCOMM'09*, pages 63–74. ACM, Aug 2009.

[5] Yantao Sun, Xiaoli Song, Bin Liu, Qiang Liu, and Jing Cheng. Matrixdcn: A new network fabric for data centers[online]. `http://tools.ietf.org/html/draft-sun-matrix-dcn-00`, 2012.

[6] John T Moy. *OSPF: anatomy of an Internet routing protocol*. Addison-Wesley Professional, 1998.

[7] Dan Li, Mingwei Xu, Hongze Zhao, and Xiaoming Fu. Building mega data center from heterogeneous containers. In *19th IEEE International Conference on Network Protocols (ICNP)*, pages 256–265. IEEE, Oct 2011.

[8] Joe Touch and Radia Perlman. Transparent interconnection of lots of links (trill): Problem and applicability statement[online]. `http://tools.ietf.org/html/rfc5556`, 2009.

[9] Cisco. Fabricpath [online]. `http://www.cisco.com/en/US/netsol/ns1151/index.html`.

[10] Changhoon Kim, Matthew Caesar, and Jennifer Rexford. Floodless in seattle: a scalable ethernet architecture for large enterprises. In *ACM SIGCOMM'08*, pages 3–14. ACM, Aug 2008.

[11] Jayaram Mudigonda, Praveen Yalagandula, Mohammad Al-Fares, and Jeffrey C Mogul. Spain: Cots data-center ethernet for multipathing over arbitrary topologies. In *Proceedings of the 7th USENIX conference on Networked systems design and implementation*, pages 18–18. USENIX Association, Mar 2010.

[12] Jayaram Mudigonda, Praveen Yalagandula, Jeff Mogul, Bryan Stiekes, and Yanick Pouffary. Netlord: a scalable multi-tenant network architecture for virtualized datacenters. In *ACM SIGCOMM'11*, pages 62–73. ACM, Aug 2011.

[13] Dave Katz and Dave Ward. Bidirectional forwarding detection(bfd) for ipv4 and ipv6 (single hop))[online]. `http://tools.ietf.org/html/rfc5881`, 2010.

[14] Radhika Niranjan Mysore, Andreas Pamboris, Nathan Farrington, Nelson Huang, Pardis Miri, Sivasankar Radhakrishnan, Vikram Subramanya, and Amin Vahdat. Portland: a scalable fault-tolerant layer 2 data center network fabric. In *ACM SIGCOMM'11*, pages 39–50. ACM, Aug 2009.