WIRELESS COMMUNICATIONS AND MOBILE COMPUTING
Wirel. Commun. Mob. Comput. 2016; **16**:942–959
Published online 18 February 2015 in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/wcm.2579

RESEARCH ARTICLE

MatrixDCN: a high performance network architecture for large-scale cloud data centers

Yantao Sun[1], Min Chen[2*], Limei Peng[3], Mohammad Mehedi Hassan[4] and Abdulhameed Alelaiwi[4]

[1] School of Computer and Information Technology, Beijing Jiaotong University, Beijing, China
[2] School of Computer Science and Technology, Huazhong University of Science & Technology, Wuhan, China
[3] Department of Industrial Engineering, Ajou University, Gyeonggi-do, South Korea
[4] College of Computer and Information Sciences, King Saud University, Riyadh, Saudi Arabia

ABSTRACT

With the widespread deployment of cloud services, data center networks are developing toward large-scale, multi-path networks. Conventional switching-oriented data center network meets difficulties in terms of scalability and flexibility to support increasing bandwidth requirements for cloud services. To solve this problem, a simple and scalable architecture, MatrixDCN, is proposed in this paper. MatrixDCN is an approximate non-blocking network, in which switches and servers are arranged in rows and columns that compose a matrix structure. A MatrixDCN network can accommodate up to hundreds of thousands of servers without bandwidth bottlenecks. Furthermore, the physical topology of a MatrixDCN network can be designed consistently with its logic topology, which helps to reduce the complexity of the management and maintenance of a data center. An efficient routing algorithm, named fault-avoidance routing (FAR), is well designed for MatrixDCN to fully leverage the regularity in the topology. FAR builds two routing tables for a router. A BRT is built based on local topology, and a novel negative routing table (NRT) is increasingly built based on learned partial network failures, which really avoids the problem of network convergence and further shortens the calculating time of routing tables. FAR also greatly reduces the size of routing tables by introducing NRTs at routers. Theoretical analysis and simulations show that MatrixDCN has advantages on the scalability of topology, network throughput, and the performance of FAR.

KEYWORDS
data center network; network architecture; routing method

***Correspondence**
Min Chen, School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan, 430074, China.
E-mail: minchen@ieee.org

1. INTRODUCTION

With the rapid development of cloud computing technologies [1], widely deployed cloud services such as Amazon EC2 (Amazon.com, Inc., Seattle, WA, USA) and Google search (Googleplex, Mountain View, CA, USA) bring about huge challenges to data center networking (DCN)[†].

A DCN is a facility used to house computer systems and associated components such as telecommunications and storage systems[wiki]. The cloud computing technology is to put a large number of computing and storage resources into a data center(cloud) and to draw out these resources on demands from a cloud by end users just like using electric power.

Today's data centers (DCs) require large-scale networks with higher internal bandwidth and lower transfer delay, but conventional networks cannot meet such requirements because of its limitations in their network architecture [2,3].

A layered multi-root tree architecture is commonly used in conventional data centers, in which many layer 2 domains connect together via layer 3 networks, as shown in Figure 1. A layer 2 domain consists of many servers that are divided into subnets (i.e., virtual local area networks, VLANs), and each subnet contains up to a few hundred servers connected via layer 2 switches.

The conventional architecture cannot support a large-scale data center with up to tens of thousands of servers in

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

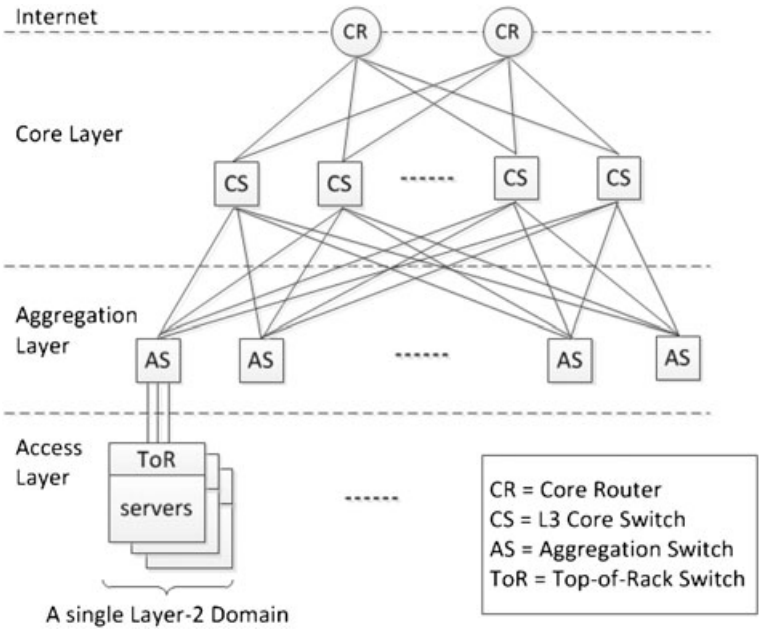


Figure 1. A conventional network for data centers.

one site [3]. The main problem that limits the scale of a DCN is a shortage of bandwidth in higher layers. In practice, the typical oversubscription ratio between neighbor layers is 1:5 or more, and at the top layer, it might reach 1:80 to 1:240 [4]. Although the most advanced and fastest switches and routers are adopted in this architecture, only 50% of the aggregate bandwidth of edge networks can be supported in the top layer [5]. It is no doubt that the top layer is becoming a bottleneck for the entire network, especially in today's cloud computing environment, when the requirement intra-network traffic is increasing rapidly.

In recent years, some new architectures have been proposed to overcome the limitation of conventional DCNs [6], such as fat-tree and BCube. Fat-tree [7] is a switch-centric network, which has better compatibility with conventional networks. A conventional data center is easier upgraded to a fat-tree network. BCube [8] is a server-centric networks, which has better scalability to support larger-scale data centers. But its network routing occupies some resource of each server, which results in decreasing performance of servers.

To accommodate tens of thousands of servers, large number of switches or servers with routing functionality are required in a DCN. For example, to accommodate 27 648 servers, a fat-tree network requires 2880 switches; each of which has 48 switching ports. It is unlikely that generic routing protocols such as Open Shortest Path First (OSPF) and Intermediate System-to-Intermediate System (IS-IS) can be scaled to support several thousands of routers [9], 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 [10]. 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 related routing protocols, which enables the reuse of the routing device when the architecture of a DCN 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 current routing protocols because the structure and query method of their routing table, fault-tolerance, and routing mechanism are all different with current routing protocols. It is 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 [7,8].

To support large-scale cloud data centers with good scalability, we propose a novel network architecture, MatrixDCN, in this paper. A MatrixDCN network is a switch-centric network and is composed of three types of switches, that is, access switches (ASes), row switches (RSes), and column switches (CSes). ASes connect servers to a network and are organized as a matrix with multiple rows and columns. RSes/CSes link ASes that lie in a row/column together.

Furthermore, to solve several problems on routing of DCNs mentioned previously, including adaptability, efficiency, and complexity of routing methods, we propose a routing method, named fault-avoidance routing (FAR) method, for MatrixDCN. Besides a common routing table, named basic routing table (BRT), FAR introduces a novel routing table, named negative routing table (NRT). The structure of a NRT is the same as a BRT, but its route entries present avoiding routes that pass through some failed links. FAR looks up both a BRT and NRT to decide a route for incoming packets. FAR is also a generic routing method. By slight revises, it can be used in many kinds of network architectures with regular topologies.

As opposed to the existing DCN architectures, MatrixDCN has the following advantages:

- MatrixDCN can support large-scale networks without bandwidth bottlenecks. With the same network performance, MatrixDCN can support 4/3 to two times more servers than fat-tree in a data center.
- The physical topology of a MatrixDCN network can be designed consistently with its logic topology, which helps to reduce the complexity of management and maintenance in a DCN.
- FAR builds BRTs based on local topology and increasingly builds NRTs based on learned partial network failures, which 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 large-scale DCNs.
- By introducing NRT, FAR decreases the size of routing tables. In a DC, which contains tens of thousands of severs, a FAR routing table only requires tens of route entries.

The main contributions of this paper are two-fold: (i) introduce a new non-tree network architecture for large-scale DCNs and (ii) propose a new high-efficient routing method for regular network topologies.

The remainder of this paper is organized as follows. Section 2 introduces the state-of-art research studies on DCN architectures and related routing methods. Section 3 presents the structure and addressing scheme of MatrixDCN. Section 4 introduces FAR, including its framework, the construction of routing tables, and its routing procedure. Section 5 introduces how to deploy a MatrixDCN network. Section 6 analyzes the the performance of MatrixDCN on its scalability, network throughput, and the performance of FAR. Section 7 verifies the performance of MatrixDCN through OPNET simulations (Riverbed Technology, San Francisco, CA, USA). Section 8 concludes this paper.

2. RELATED WORK

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

Fat-tree [7] is a typical switch-centric network. In 2008, M. Al–Fares *et al.* proposed the fat-tree network on the conference of Special Interest Group on Data Communication (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. Using 48-port switches, a fat-tree network can accommodate up to 27 648 servers. To support non-blocking communication, the fat-tree architecture 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 this problem, VL2 [4] replaces cheap core and aggregation switches in fat-tree with expensive high-speed electrical switches.

DCell [11] and BCube [8] 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 cause traffic jam in network. To solve traffic jam in DCell, 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 100 000 of servers. FiConn [12] shares the same design principle as DCell, but limits a server node's degree to two. DPillar is built of two types of devices, n-port switches and dual-port servers [13]. Visually, it looks like the 2 k 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 called Bidimensional Compound Networks (BCN) for compound graphs [14]. Compared with the structures previously, BCN is more complicated and cannot eliminate traffic bottleneck in a network.

Some studies focus on irregular topologies. Scafida [15] is inspired by the scale-free Barabási and Albert topologies. Curtis *et al.* proposed ReWire [16] that is a framework to design, upgrade, and expand DCNs. 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.

Some new communication technologies were introduced to DCN. To reduce networking cost, optical circuit switches are used in Helios [17] and c-Through [18] to build a hybrid electrical/optical architecture. Wireless technology is also used in DCNs. In 2011, Daniel Halperin *et al.* proposed adding multi-gigabit wireless links to the wired DCN to relive network congestion and thus to improve performance [19]. In 2012, Xia Zhou *et al.* proposed a new wireless primitive for data centers, 3D beamforming, where 60 GHz signals bounce off data center

ceilings thus establishing indirect line-of-sight between any two racks in a data center [20].

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

Some generic routing methods for DCNs have been proposed such as transparent interconnection of lots of links (TRILL) [21], FabricPath [22], and Seattle [23]. The aim of these routing methods is mainly to solve the problem of layer 2 inter-connection of DCNs. As the basis of these routing method is still the link-state routing protocols, that is, OSPF or Intermediate System-to-Intermediate System, they are also not applicable to a large-scale DCN with tens of thousands of severs.

Smart Path Assignment In Networks (SPAIN) [24] and NetLord [25] demonstrate a new thinking about layer 2 interconnection based on existing switch devices in an arbitrary topology. Within these methods, a set of paths are pre-computed off-line 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 on 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 dedicated routing algorithms for some network architectures. These routing algorithms leverage the regularity in the topology, so they have higher efficiency on route computing and packet forwarding.

Several routing solutions are proposed for fat-tree networks. The paper [7] proposes two-level routing to spread outgoing traffic on multiple equal cost paths. When making a routing decision, a switch looks up a main routing table first. If no route is hit, then the switch looks up a small secondary table. To implement fault-tolerance in a routing procedure, fat-tree detects link failures by a Bidirectional Forwarding Detection session [26] and broadcasts those failures between switches. The paper [27] presents a dynamic local rerouting methodology for fat-tree networks. It can guarantee connectivity for up to and including $k - 1$ benign faults using either deterministic or adaptive routing, where k is half the number of ports in the switches. PortLand [28] is a layer 2 network solution specifically for fat-tree network. It uses a lightweight location discovery protocol that allows switches to discover their location in the topology. In PortLand, every end host is assigned an internal pseudo Mac that encodes the location of the end host. 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.

BCube is a sever-centric network architecture in which servers are responsible for both computing service and routing function. In BCube [8], a source routing protocol called BCube Souce Routing (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.

ALIAS [29] is an addressing and communication protocol that automates topology discovery and address assignment for the hierarchical topologies. ALIAS first assigns hierarchical, topologically meaningful labels to hosts and then enables communication over these labels. ALIAS has good fault-tolerance. It dynamically recovers from arbitrary failures, without requiring modifications to hosts or to commodity switch hardware.

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 as follows: 1) they are closely designed for the special topology of networks and 2) their fault-tolerance mechanisms are complicated.

3. MATRIXDCN ARCHITECTURE

We propose a novel network architecture, MatrixDCN, for large-scale data centers with tens of thousands of servers. Its structure is quite simple and clear and has good scalability and flexibility.

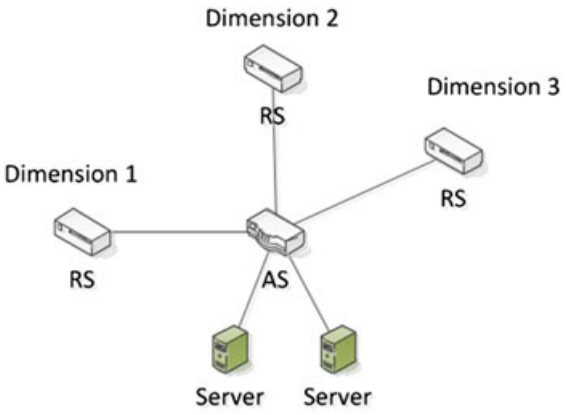


Figure 2. An access switch (AS) in a 3D MatrixDCN network.

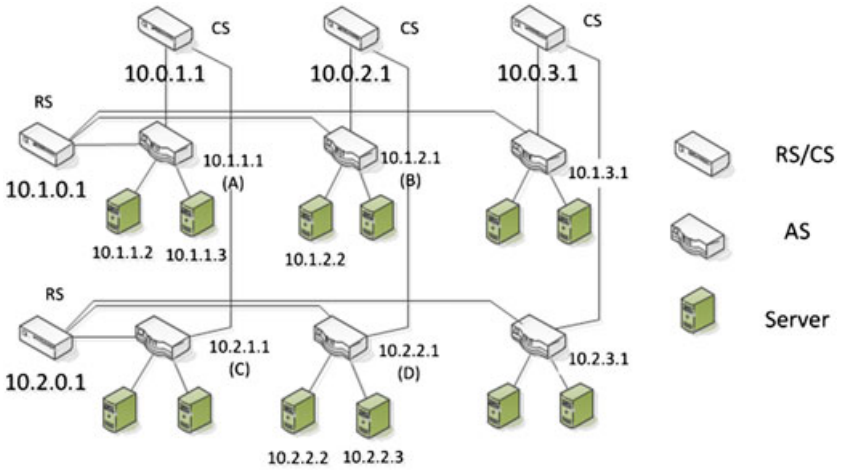


Figure 3. A 2 × 3 MatrixDCN network.

3.1. Multi-dimension network

In a MatrixDCN network, ASes[‡] are deployed as a logical multi-dimensional matrix, and each AS connects a group of servers, so a MatrixDCN network can accommodate a huge number of servers. For example, a $40 \times 40 \times 40$ 3D MatrixDCN accommodates 64 000 ASes, and the network has 640 000 servers if an AS connects only 10 servers.

In each dimension, a MatixDCN network can be viewed as many rows of ASes. Each row deploys with a group of RSes, and each RS connects all of the ASes together in a row. In other works, an AS connects to the RSes of the same row in each dimension. For example, in a $d1 \times d2 \times d3$ 3D network, there are $d2 \times d3$ rows, and each row has $d1$ ASes in the first dimension. Figure 2 shows an AS that connects to the RSes in each dimension in a 3D network.

A 2D MatrixDCN network can satisfy the requirement of scale in network capacity for most of real scenes. For convenience, we call the rows and RSes in dimension 2 as columns and CSes, respectively, in a 2D MatrixDCN network. Figure 3 shows an example of a 2×3 MatrixDCN. A 2D MatrixDCN is a very flexible network structure and can be deployed with any number of rows/columns and any number of RSes/CSes in one row/column according to the requirements of network scale and bandwidth, in practice.

Like BCube, MatrixDCN can be used to build an extra-large-scale DCN. Suppose that all of the switches are N-port switches, then a k-dimensional MatrixDCN network can support N^k ASes. If the ports of an AS are divided into $k + 1$ equal parts to connect servers and RSes in each dimension, respectively, then each AS links $N/(k + 1)$ servers, and each row has $N/(k + 1)$ RSes. Thus, the network can accommodate a maximum of $N^{(k+1)}/(k + 1)$ servers and requires $kN^k/(k + 1)$ RSes. The ratio of servers to switches is $N/(2k + 1)$. If the network has two dimensions, the ration is $N/5$, which is equal to that of fat-tree.

In a 2D MatrixDCN network, if the bandwidth between an AS and its connected servers (access bandwidth, AB) is equal to the bandwidth between the AS and its connected RSes (row bandwidth, RB) and the bandwidth between the AS and its connected CSes (column bandwidth, CB), that is, $AB = RB = CB$, the MatrixDCN network is an approximate non-blocking network. Generally, in a k-dimensional MatrixDCN network, if AB is equal to RB in each dimension, the network is an approximate non-blocking network.

If it is known in advance that most traffic is the local traffic whose source and destination lie in the same subnet, then less RB and CB is required; more ports of an AS can be used to link servers, and more servers can be accommodated in a network. Besides, most commodity switches have several uplink ports that have a higher speed for switch cascading. In general, the speed ratio of uplink ports to downlink ports is 10:1. For an AS, if its uplink ports are used to link CSes and RSes, and its downlink ports are all used to link servers, more servers can be accommodated in a MatrixDCN network. In this case, RSes and CSes should be replaced with faster switches to match the speed of ASes' uplink ports.

3.2. Addressing in MatrixDCN

As shown in Figure 3, each device's IP address is configured according to its location in a 2-dimensional MatrixDCN network. The Internet Protocol (IP) addresses of RSes, CSes, and ASes are set to 10.*rno*.0.*x*, 10.0.*cno*.*x* and 10.*rno*.*cno*.1, respectively. The IP addresses of servers are set to 10.*rno*.*cno*.*x*($x > 1$). The *rno* and *cno* are the row number and column number of a device in the matrix.

As length of IPv4 address is too short, there is a trouble when applying the addressing method previously to a k-dimensional MatrixDCN network. One solution is to adapt longer IPv6 addressing method. Another solution is using

[‡]AS, RS, and CS are the singular forms of ASes, RsSes and CSes, respectively.

small segment of IPv4 address to indicate a row number. For example, in a 3D MatrixDCN network, three row numbers and one server number share the last 24 bits in an IPv4 address, so a number only occupy 6 bits other than 1 bytes in a 2D network.

Encoding the location and type of a device into its IP address helps to determine the adjacency between the devices and further simplifies the network routing scheme. In addition, that each switch is assigned to only one IP address greatly simplifies the network configuration and management.

4. FAR ROUTING

A distributed routing method, FAR, is proposed for MatrixDCN. FAR simplifies route computing by leveraging the regularity in network topologies and solves the problem of lack of adaptability in existing routing methods for DCNs. Compared with other routing methods, FAR is simpler and more efficient and has good fault-tolerance performance.

FAR has good adaptability. It can be used in many kinds of network topologies by slight revises. FAR only requires that a DCN 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 DCNs. To run FAR in a DCN, each router or switch with routing functionality should be deployed with a FAR instance.

4.1. The framework of FAR

FAR is divided into three parts: 1) a link state learning unit that learns all the link failures in the entire network; 2) a routing table building unit that builds up routing tables according to learned link states; and 3) a routing table querying unit that looks up routing tables to forward packets. The framework of FAR is shown in Figure 4.

The link state learning unit consists of four 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 A and its neighbor B breaks, then B cannot be detected by A through a heartbeat mechanism. In this case, B is called an invisible neighbor of 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 BRT and a 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, 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 links learned in M4. Routing table querying module (M7) looks up both the NRT and BRT and then combines query results together to forward incoming packets.

4.2. Limited state machine of FAR

Figure 5 shows how a FAR router works by its finite state machine(FSM). There are 16 key steps in the FSM:

- Step 1: When a router starts up, it starts a hello thread and then starts ND (neighbor detection) timer (3 s). Next, the router goes into ND state.
- Step 2: In the ND state, if a router received a hello message, then it performs a hello-message processing and goes back to the ND state.
- Step 3: When the ND timer is over, a router goes into ND-FIN (neighbor detection finished) state.
- Step 4: A router starts the LFD (link failure detection) thread and DFD (device failure detection) thread and sends device announcement (DA) and device link request (DLR) messages to all of its active ports. Then, the router goes into listen state.

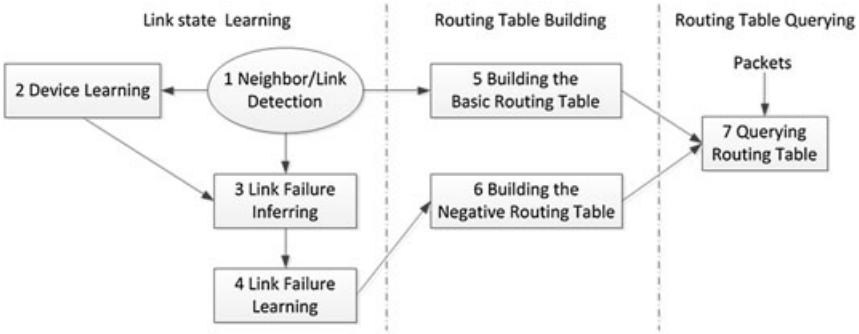


Figure 4. The routing framework of FAR.

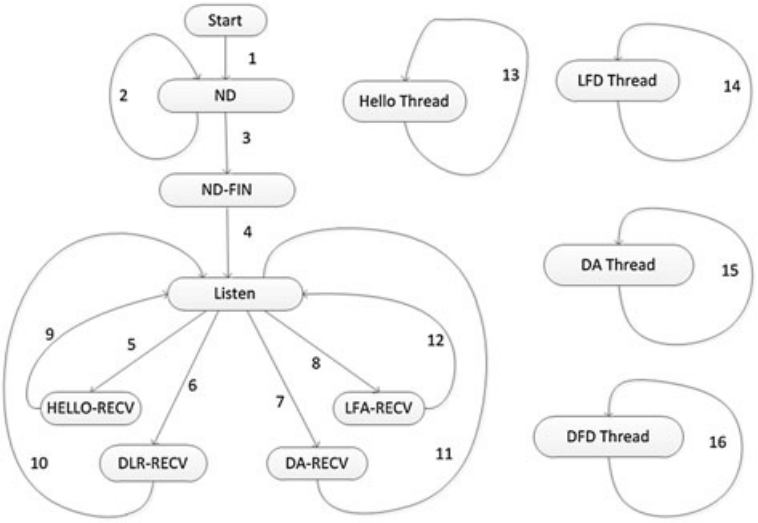


Figure 5. Finite state machine of FAR.

- Step 5–8: If a router receives a message then goes into a corresponding state to process the received message.
- Step 9–12: After a router performs a message processing, it goes back to listen state.
- Step 13: Hello thread produces and sends hello messages to all its ports periodically.
- Step 14: LFD thread calls link-failure-detection processing to check link failures in all links periodically.
- Step 15: DA thread produces and sends DA messages periodically (30 min).
- Step 16: When DFD thread starts up, it sleeps a short time (30 s) to wait for a router learning all the active routers in the network. Then, the thread calls the device-failure-detection processing to check device failures periodically (30 min).

4.3. Learn link states

To avoid the failed links when forwarding packets, each switch needs to know the states of all links, active or failed. FAR spreads all the switches and link-state changes in the network 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 because 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 the received hello messages and stores its neighbors and links in a local database.

(2) Device Learning
FAR learns all the switches in the entire network through DA and 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. DAs will be repeatedly published by a long period such as 30 min.

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 10.1.1.1 fails, so it can't be detected by node 10.1.0.1 through the heartbeat mechanism. But according to the regularity in topology and assignment rules of IP address in MatrixDCN, 10.1.0.1 can infer that there exists a failed AS whose IP address is 10.1.1.1, and the AS 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 CS 10.0.1.1 (S_a), and AS 10.1.1.1 (S_b) are adjacent. If S_a learns the existence of S_b through a DA message but S_a cannot 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 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. Besides a DA message, M4 answers an LFA message that encapsulates all the link failures in its database for response. M4 also broadcasts it local link failures to its neighbors through an LFA message. When M4 receives an LFA message, it stores unknown link failures encapsulated in the message into its local database and then forwards them to its neighbors except for the incoming one.

4.4. Basic routing table

By leveraging the regularity in topology, FAR calculates routes for any destination and builds a BRT for a router through local topological knowledge. M5 module builds up a BRT for a router depending on detected neighbors and links in M1.

In 2D MatrixDCN, RSes/CSes are responsible for forwarding packets between columns/rows, respectively, and ASes decide whether a packet is to be forwarded to an RS, a CS, or the destination server directly. Communication between servers within a subnet only requires the AS' participation of that subnet. When servers lie in the same row/column, their communication requires the participation of the RSes/CSes in that row/column besides ASes. For example, the communication path between the servers 10.1.1.2 and 10.1.2.2 in Figure 3 is: 10.1.1.2 $\rightarrow$ A(10.1.1.1) $\rightarrow$ RS(10.1.0.1) $\rightarrow$ B(10.1.2.1) $\rightarrow$ 10.1.2.2. The most complex communication between the servers appears in different rows and columns. This type of path requires the participation of RSes, CSes, and ASes. For example, if server 10.1.1.2 communicates with 10.2.2.2 in Figure 3, there are two groups of routing paths. One group is CS-first: 10.1.1.2 $\rightarrow$ A $\rightarrow$ CS $\rightarrow$ C $\rightarrow$ RS $\rightarrow$ D $\rightarrow$ 10.2.2.2. The other group is RS-first: 10.1.1.2 $\rightarrow$ A $\rightarrow$ RS $\rightarrow$ B $\rightarrow$ CS $\rightarrow$ D $\rightarrow$ 10.2.2.2.

The routes discussed previously are described through BRTs. Different types of switches have correspondingly different BRTs. First, we discuss the BRT of an RS. If a packet with the destination address 10. $*$ .cno.$*$ arrives at RS 10.rno.0.i, the RS should forward it to the next hop, that is, the AS 10.rno.cno.1. Thus, in an $m \times n$ MatirxDCN, the BRT of RS 10.rno.0.i looks like the following:

Destination/mask	Next hop
10.0.1.0/255.0.255.0	10.rno.1.1
10.0.2.0/255.0.255.0	10.rno.2.1
...	
10.0.n.0/255.0.255.0	10.rno.m.1

Because MatrixDCN is a symmetrical architecture, the BRT of a CS is similar to that of an RS. Here, we do not discuss BRTs of CSes. Next, we discuss the BRT of an AS. If a packet with destination 10.r2.c2.$*$ arrives at AS 10.r1.c1.1, there are four cases. 1) $r1 = r2$ and $c1 = c2$, which means that the packet arrives at the destination subnet; thus, the AS will directly send the packet to the destination server through layer 2 switching. 2) $r1 = r2$, $c1 \neq c2$, which means that the destination server lies in the same row with the AS; thus, the AS will forward the packet to an RS 10.r1.0.$*$. 3) $r1 \neq r2$, $c1 = c2$, which means that the destination server lies in the same column with the AS; thus, the AS will forward this packet to a CS 10.0.c1.$*$. 4) $r1 \neq r2$ and $c1 \neq c2$, which means that the destination lies in a different row and column; thus, the AS will forward this packet to anyone of the RSes 10.r1.0.$*$ or CSes 10.0.c1.$*$. So the BRT of AS 10.r1.c1.1 looks like the following:

Destination/mask	Next hop
10.r1.0.0/255.255.0.0	10.r1.0.$*$
10.0.c1.0/255.0.255.0	10.0.c1.$*$
10.0.0.0/255.0.0.0	10.r1.0.$*$
10.0.0.0/255.0.0.0	10.0.c1.$*$

In the table previously, 10.r1.0.$*$ represents the set of RSes in row $r1$. Therefore, each line denotes a group of route entries, and each RS corresponds to one route entry. Similarly, 10.0.c1.$*$ represents the set of CSes in column $c1$.

Now, we discuss the computational complexity of a BRT. Because a router calculates its BRT entries only using the information of neighbor nodes and independent of the scale of a network, the calculation consumes constant time and memory; it can be deduced that the time and space complexity of calculating route entries for a neighbor node are both O(1). The total time complexity for all neighbor nodes is O(N), where N is the number of neighbor nodes. Because we do not need additional memory, the total space complexity is still O(1).

The BRT in k-dimensional MatrixDCN ($k > 2$) is very similar to 2D MatrixDCN, so we do not discuss it in this paper.

4.5. Negative routing table

Because link failures are not considered while calculating a BRT, the routes in BRTs cannot avoid failed links. There are two methods to solve this problem. One is to tell a switch what hops, that is, paths; it can forward packets, and another one is to tell a switch what hops; it cannot. Conventional routing methods such as OSPF and Routing information Protocol (RIP) are all based on the first method. In FAR, the second one is used. FAR uses an NRT to exclude the routing paths that pass through some failed links from the paths calculated by BRTs. The M6 module builds up an NRT for a router depending on its learned link failures in M4.

Compared with conventional routing method, using an NRT to avoid failed links in a multiple-path network is simple and can decrease the size of routing tables remarkably. For example, in Figure 6, if the network has no failures, the routing table of node 10.1.48.1 has 16 route entries as follows:

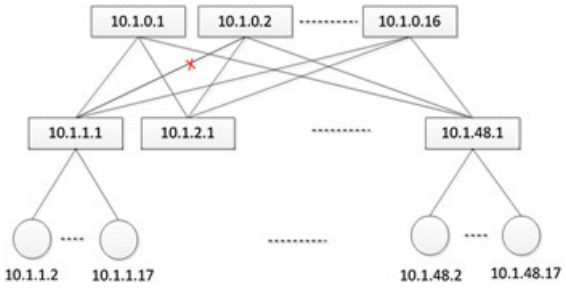


Figure 6. An example of multiple paths network.

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 fail, in order to avoid this failed link for communication from subnet 10.1.48.0/24 to subnet 10.1.1.0/24, in conventional routing methods, node 10.1.48.1 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 previous entries tell the switch that the traffic to subnet 10.1.1.0/24 can be forwarded to 10.1.0.1, 10.0.3, ..., and 10.1.0.16 except for 10.1.0.2.

But in FAR, only one route entry 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.

The M6 module builds up its NRT for a switch depending on the link failures learned in M4. Next, we illustrate how a switch builds its NRT in a 2D MatrixDCN network. In MatrixDCN, only an AS requires an NRT. NRT entries are related to the location of link failures.

We divide all the links in a 2D MatrixDCN network into three catalogs: row links, column links, and access links. Row links connect ASes and RSes; column links connect ASes and CSes; and access links connect ASes and servers. We explicitly call the links connected to an AS as the links of the AS.

We use A_{ij} to denote the AS at the cross position of the *ith* row and the *jth* column. R_i denotes all of the RSes in the *ith* row. C_i denotes all of the CSes in the *ith* column. $R(A_{ik})$ and $C(A_{ik})$ denote the set of RSes and CSes that the failed links of A_{ik} connect to, respectively.

The location of link failures affects routing decision of ASes. For AS A_{ij}, there are four cases of link failures: 1) some of links of A_{ij} fail; 2) some of links of another AS A_{ik} in the same row fail; 3) some of links of another AS A_{lj} in the same column fail; and 4) some of links of another AS A_{lm}, which lies in a different row and column, fail.

For case 1, A_{ij} will directly remove route entries related to failed links from its BRT. For example, if link $A_{ij} \leftrightarrow RS(10.i.0.1)$ fails, the following two entries should be removed from the BRT of A_{ij}.

Destination/mask	Next hop
10.*i*.0.0/255.255.0.0	10.*i*.0.1
10.0.0.0/255.0.0.0	10.*i*.0.1

Cases 2 and 3 are two symmetrical cases, so we only discuss case 2. In case 2, if some of row links of A_{ik} fail, they will affect the routing of the packets whose destination lie in the *kth* column because these packets will pass through A_{ik}. So we should remove the paths relevant to those failed links. The paths to be avoided are stored in the NRT of A_{ij} as follows:

Destination/mask	Next hop
10.0.*k*.0/255.0.255.0	$R(A_{ik})$

The previous route entry means that $R(A_{ik})$ should be excluded from the next hops for the destination 10.0.*k*.0/255.0.255.0. We call such a route entry a negative route entry.

In case 2, if a part of the column links of A_{ik} fails, these failed links will not affect routing of A_{ij}. But if all of the column links of A_{ik} fail, it is obvious that A_{ik} cannot act as an intermediate transmitting node between A_{ij}, and the nodes lied in the *kth* column; thus, all of the RSes on the *ith* row should be excluded from the next hops with the destination 10.0.*k*.0/255.0.255.0, which is expressed in the NRT of A_{ij} as follows:

Destination/mask	Next hop
10.0.*k*.0/255.0.255.0	R_i

For case 4, if a part of the row or column links of A_{lm} fails, these failed links will not affect routing of A_{ij}. But if all row links of A_{lm} fail, to avoid packets passing through the row links of A_{lm}, A_{ij} should not transfer packets whose destination is A_{lm} to its row links. So, the following entries should be added to the NRT of A_{ij}:

Destination/mask	Next hop
10.*l*.*m*.0/255.255.255.0	C_j

Similarly, if all the column links of A_{lm} fail, the following entries should be added to the NRT of A_{ij}:

Destination/mask	Next hop
10.*l*.*m*.0/255.255.255.0	R_i

By means of the rules previously, FAR can increasingly build up an NRT based on partial link failures that a router has learned, without waiting for the network to approach the state of convergence.

Now, we discuss the computational complexity of a NRT. An AS calculates NRT entries based on received link failures. When an AS receives a link failure, it performs one time of calculating NRT entries, and the calculation consumes constant time and space. So the time and space complexity of NRT entries are both O(1) for a link failure.

The NRT in k-dimensional MatrixDCN ($k > 2$) is very similar to 2D MatrixDCN, so we do not discuss it in this paper.

4.6. Lookup routing tables

FAR looks up both a BRT and an 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. Second, 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. Third, a set of applicable entries is composed of the candidate entries minus the avoiding 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 a routing decision procedure. In this example, we suppose that the link between 10.1.0.1 and 10.1.2.1 has failed in a 2D MatrixDCN network. Next, we look into how node 10.1.1.1 forwards a packet to the destination 10.2.2.2.

(1) Calculate candidate hops. 10.1.1.1 looks up its BRT and obtains the following matched entries:

Destination/mask	Next hop
10.0.0.0/255.0.0.0	10.1.0.1
10.0.0.0/255.0.0.0	10.0.1.1

So the candidate hops = {10.0.1.1, 10.1.0.1}.

(2) Calculate avoiding hops. 10.1.1.1 looks up its NRT and obtains the following matched entries:

Destination/mask	Next hop
10.0.0.0/255.0.0.0	10.1.0.1

So the avoiding hops = {10.1.0.1}.

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

(4) Finally, 10.1.1.1 forwards the packet to the next hop 10.0.1.1.

5. DEPLOYMENT

As a 2D MatrixDCN network is big enough to support the requirement of scale in DCN, we just introduce how to deploy a 2D MatrixDCN network in this section. The deployment of a 2D MatrixDCN network is very easy. In 2D networks, switches and servers are deployed on racks, and racks are arranged in rows. An AS and its connected servers are deployed on a server rack; RSes in a row and CSes in a column are deployed on an RS rack and a CS rack, respectively. Devices between racks are connected together via twisted-pair wires or optical cables. All row links of an AS are bundled together to connect each RS in the same row. Correspondingly, all column links of an AS are bundled together to connect each CS in the same column. As shown in Figure 7.

In Figure 7, only two server racks, two CS racks, and one RS rack are depicted, and other racks are omitted. Figure 7 shows that physical deployment of devices in a MatrixDCN network is consistent with logical topology of the network, which is helpful to reduce the complexity of network management and maintenance.

6. NETWORK PERFORMANCE ANALYSIS

In this section, we discuss the performance of MatrixDCN from the perspectives of network throughput, scalability, and the performance of FAR.

6.1. Network throughput

To support cloud services, more and more internal bandwidth is required in today's data centers, and the best situation is to have no bandwidth bottleneck or traffic blocking anywhere in a data center. Similar to the Fat-tree architecture, MatrixDCN can eliminate bandwidth bottlenecks and can achieve an approximate non-blocking network. In this section, we take 2D MatrixDCN as an example to analysis MatrixDCN's performance on network throughput.

Obviously, traffic blocking occurs only on row links and column links in a 2D MatrixDCN network. If a 2D network is configured as $RB = CB = AB$, the bisection bandwidth is equal to *the access bandwidth*×*the number of servers*/2; thus, it is possible to eliminate a traffic blocking in the network through appropriate traffic arrangements. In fact, such MatrixDCN network is an approximate non-blocking network in which only a small amount of traffic can be blocked with a low probability. Next, we calculate the blocking probability and blocked volume of traffic in a MatrixDCN network. Because MatrixDCN is asymmetrical structure, the traffic distribution on row links is the

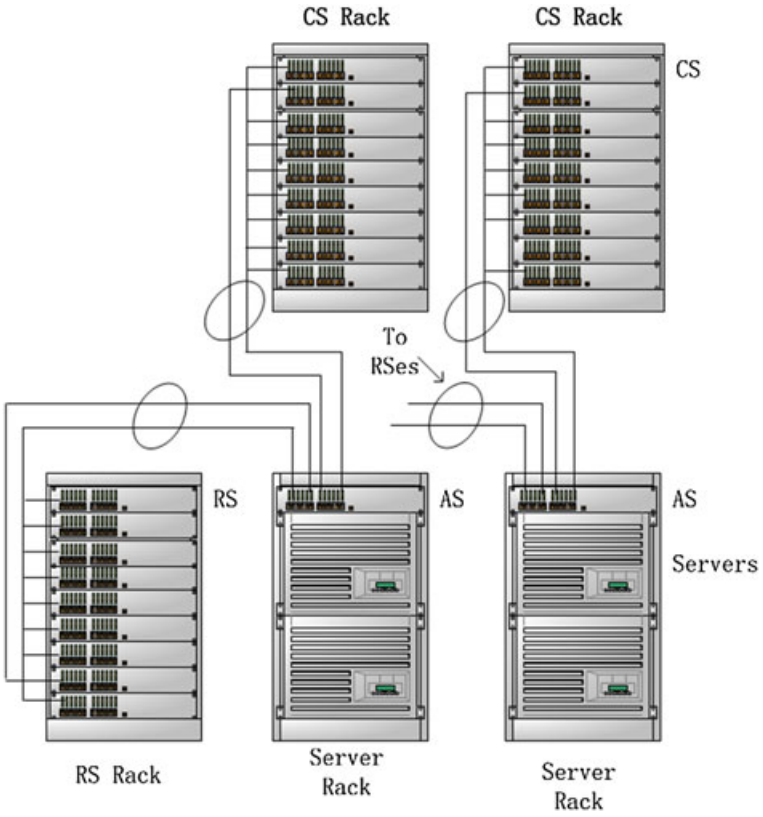


Figure 7. Deployment of a MatrixDCN network.

same as on column links. Here, we calculate only the traffic in column links. In addition, links are bidirectional, and also the traffic of two ways are symmetrical, so we only calculate the traffic of one way over column links.

We suppose that the scale of a MatrixDCN network is m rows $\times$ m columns, where each AS connects with K servers and $RB = CB = AB = K$ for an AS. Each server consumes one unit of bandwidth. To simplify calculation, we specify that an AS first forwards a packet to CSes if the packet passes through a different row and column. For the column links of A_{ij}, only when the servers in the *jth* column communicate with the servers in the *ith* row will the traffic pass through these links from CSes to A_{ij}. Suppose that traffic are uniformly distributed, so the probability that a server communicates with the servers in the *ith* row is $1/m$, and the number of servers that communicate with the servers in the *ith* row follows a binomial distribution. We use S_{ij} denoting the subnet at the cross position of the *ith* row and *jth* column. There are $(m-1)K$ servers in the *jth* column besides the servers in the subnet S_{ij}; as a result, the probability that k servers of those $(m-1)K$ servers communicate with the servers in the *ith* row is as follows:

$$P(X=k) = \binom{(m-1)K}{k} \left(\frac{1}{m}\right)^k \left(1-\frac{1}{m}\right)^{(m-1)K-k} \tag{1}$$

If more than K servers in the *jth* column communicate with the servers in the *ith* row, the traffic in the direction from CSes to A_{ij} will exceed the capacity of the row links; as a result, the excess traffic will be blocked. The blocking probability is as follows:

$$\rho = 1 - \sum_{k=0}^{K} P(X=k) \tag{2}$$

The expected value of the blocked bandwidth is:

$$B = \sum_{k=K+1}^{(N-1)K} (k-K)P(X=k) \tag{3}$$

If $m = 48$ and $K = 40$, then it can be calculated that $\rho = 39.5\%$ and $B = 2$. The ratio of the maximum network throughput, that is, aggregate throughput, to total traffic sent by all servers is $40/42 = 95.24\%$. And in this case, the total traffic is equal to the theoretical bisection bandwidth. This result is well matched with the results of our simulations.

6.2. Network scalability

MatrixDCN has a good scalability: 1) a MatrixDCN network can support a large-scale DCN with tens to thousands of servers. 2) Its scale can be easily expanded or shrunk.

Table I. Comparison of network scales.

Network type	Num of switches	Num of servers
MatrixDCN 1	3840	36 864
Fat-tree 1	2880	27 648
MatrixDCN 2	2688	110 592
Fat-tree 2	1440	55 296

As shown in Table I, with 48-port switches, a 2D MatrixDCN network (MatrixDCN 1) can support maximally 36 864 servers; and oppositely, a fat-tree network (fat-tree 1) only supports 27 648 servers. MatrixDCN 1 supports 4/3 times more servers than fat-tree 1. If we use the uplink ports of ASes to connect RSes and CSes, and all the 48 downlink ports of an AS are used to connect servers, a 2D MatrixDCN network (MatrixDCN 2) can support 110 592 servers. Correspondingly, such fat-tree network (fat-tree 2) can only support 55 296 servers, which is half of the number that a MatrixDCN can support.

In MatrixDCN 2, *RB* and *CB* are 10 times larger than *AB*, each AS has eigjt uplink ports and 48 downlink ports; each RS/CS has 48 ports, and each row/column deploys with four RSes/CSes. In fat-tree 2, the bandwidth of the other links is 10 times larger than the link between edge switches and servers; each edge switch has four uplink ports and 48 downlink ports; each core/aggregate switch has 48 ports, and each pod has four aggregate switches.

It is easy to expand or shrink the scale of a MatrixDCN network. To expand a MatixDCN network, more rows or columns can be dynamically deployed according to the real requirement. Oppositely, to shrink a MatrixDCN network, some rows and columns can be removed from the network.

6.3. The performance of FAR

We evaluate the performance of FAR with respect to the number of control messages, the route calculating time and the size of routing tables. A test MatrixDCN network is a 2D network composed of 3840 48-port switches and 36 864 servers is used in this section.

(1) The number of control messages
FAR exchanges a few messages between routers and only consumes a little network bandwidth. Table II shows the required messages in the test MatrixDCN network.

The last column of Table II presents the bandwidth consumed by each type of message on a link. From Table II, we can see that hello messages are only exchanged among neighbor switches, so hello messages use less than 4 kbps bandwidth on a link. 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 $\textit{the number of switches} \times \textit{the size of DA message} = 3840 \times 48 \times 8\,\textit{bits} \approx 1.47\,\textit{Mbits}$. Because a DA period is very long (30 min), DA messages use little bandwidth.

It can be concluded that even in a very large data center with 36 864 servers and 3840 switches, FAR produces a few number of messages and uses little bandwidth.

(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 increasingly calculated according to the learned 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. The detection and spread time of link failures is also very short in FAR. Detection time is up to the interval of sending a hello message. In FAR, the interval is set to 100 ms, and a link failure will be detected in 200 ms. The spread time between any pair of routers is less than 200 ms. In conclusion, 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 500 ms.

In OSPF, Dijkstra algorithm is used to calculate the shortest path tree (SPT), and based on the SPT, OSPF calculates a routing table. Using Dijkstra algorithm, it spent 21 s only to calculate the SPT for the test MatrixDCN network in our experiment running on a machine with an Intel I7 2.8 GHz dual-core CPU (Intel Corporation, Santa Clara, CA, USA), 8 G memory, and Windows 7 OS (Redmond, WA, USA). Furthermore, the convergence procedure of OSPF in a large-scale network requires a much longer time than the calculating time of the SPT. In general, OSPF may require totally several minutes to calculate routing tables for a large-scale network.

(3) The size of routing tables
The size of a BRT at a switch is determined by the number of neighbor switches of the switch. In the test network,the size of a BRT at a RS, CS, and AS is only 48, 48, and 64, respectively.

Only ASes require NRTs. The size of an NRT is decided by the number and locations of link failures. In the 48×48 MatrixDCN network previously, we generate 100, 200 to 1000 link failures randomly and take 10 ASes as a sample to measure the average size of NRTs, which are plotted in Figure 8. Figure 8 shows that the size of an NRT has a linear

Table II. Required messages in a MatrixDCN network.

Message type	Scope	Size	Rate	Bandwidth
Hello	Between adjacent switches	< 48 bytes	10 messages/s	<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 (3840) in a DA period (30 min)	1.47 Mbits
LFA	In the entire network	< 48 bytes	Produce one when a link fails or recovers	48 bytes

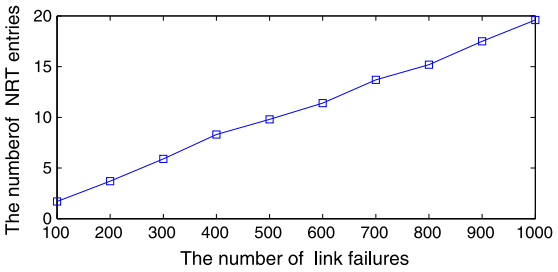


Figure 8. The size of an NRT is related to the number of link failures.

relationship with the number of link failures. When the network has up to 1000 link failures, an NRT only has about 20 entries.

If we run OSPF routing protocol in the test MatrixDCN network, each switch requires 76 032 route entries, because OSPF creates a particular route entry for each network segment in each switch, and the MatrixDCN network consists of 76 032 network segments.

For the test MatrixDCN network, the routing table sizes of FAR and OSPF are listed in Table III. Table III shows the size of routing tables in FAR is much smaller than that in conventional routing methods such as OSPF.

7. VERIFICATION BY SIMULATION

In this part, we evaluate network performance of MatrixDCN by comparing with fat-tree through OPNET simulations. We measure two key performance parameters, network aggregate throughput, and average end-to-end (ETE) communication delay. Aggregate throughput is the sum of the data rates that are delivered to all servers in a network. It indicates a network's capacity. ETE delay is the time that a packet go through a network from source node to destination node. ETE delay is an important factor of network performance.

OPNET modeler 14.5 is used in the simulation. FAR switches are developed based on the standard layer 3 Ethernet switch model in the OPNET modeler. 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.

Figure 9 is the state transition diagram of the FAR process. At the entrance of neighbor detection (ND) state, ND_timeout, and self_interrupt_hello, self-interruptions are created. At the exit of ND state, the DA and DLR messages are broadcasted to its neighbor switches. When there is an ND_timeout interruption, the 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.

We build three scenarios for both the MatrixDCN and fat-tree architectures with different sizes in our simulation. In scenario 1, we build a MatrixDCN and a Fat-tree network with 100 Mbps switches. Each network has 128 servers. The MatrixDCN network is 4×8 in which each AS links four servers, and each row/column deploys four RSes/CSes. The fat-tree network, which is built with eight-port switches, has 16 core switches and eight pods, and each pod has four aggregation switches, four edge switches, and 16 servers. In scenario 2, we upgrade link bandwidth from 100 Mbps to 400 Mbps except for the access links in both the MatrixDCN and fat-tree network. Accordingly, in the MatrixDCN network, each row/column requires only one RS/CS, and in the fat-tree network, a total of four core switches and only one aggregate switch in each pod are required. In scenario 3, we increase the sizes of the two networks and further upgrade their link bandwidth to 1000 Mbps, except for access links. The bandwidth of an access link still stays at 100 Mbps. The MatrixDCN network in scenario 3 becomes an 8×8 network with 640 servers, in which each AS links 10 servers, and each row/column deploys one RS/CS. The fat-tree network in this scenario has 10 core switches and six pods, and every pod has one aggregation switch, 10 edge switches, and 100 servers. A total of 600 servers appear in this fat-tree network.

Table III. The size of routing tables in FAR.

Routing table	Row switch (RS)	Column switch (CS)	Access switch (AS)
Basic routing table (BRT)	48	48	64
Negative routing table (NRT)	0	0	$< 20^*$
Open shortest path first (OSPF)	76 032	76 032	76 032

* The size of an NRT is decided by the number of link failures in a network, as shown in Figure 8.

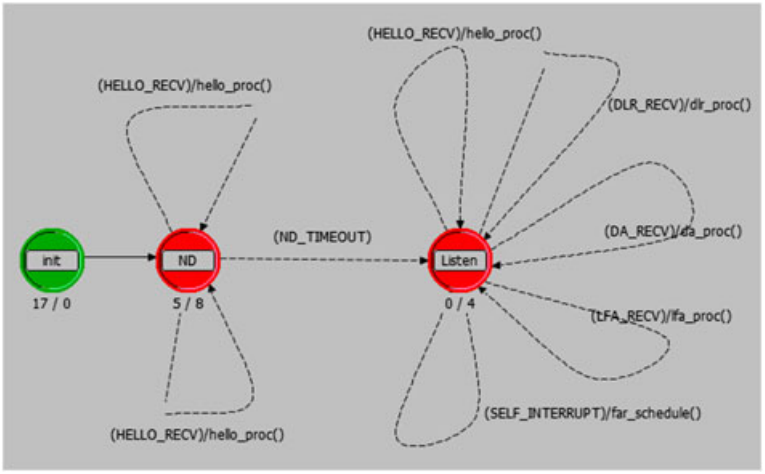


Figure 9. State transition of the FAR process.

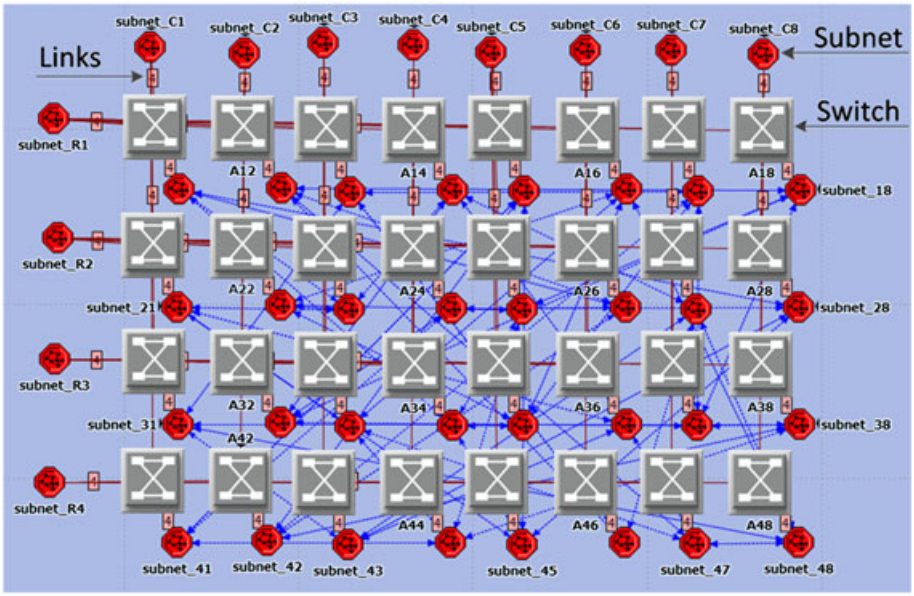
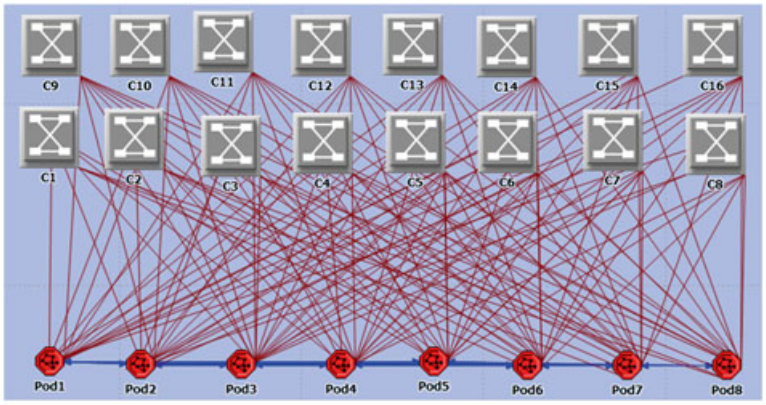
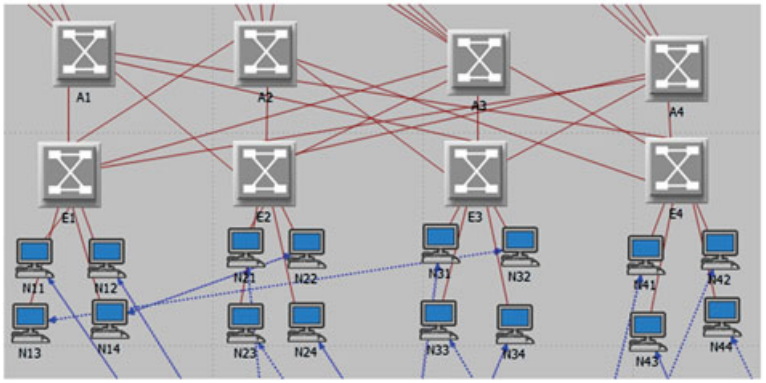


Figure 10. The MatrixDCN simulation network.



(a) The main view



(b) The view of Pod 1

Figure 11. The fat-tree simulation network.

Figure 10 is the MatrixDCN network in scenario 1. To make its topology clearer, we put all the RSes in a row, all the CSes in a column, and all the servers in a subnet into a logical subnet, respectively. Figure 11(a) is the fat-tree network in scenario 1. We put each pod of the fat-tree network into a logical subnet, as shown in Figure 11(b). In those figures, red lines represent the links between network nodes, and blue lines represent the traffic between servers. Here, we do not display the simulation networks in scenarios 2 and 3 because they are similar to scenario 1.

In our simulation, we pair off all of the servers randomly. Two servers in one pair send packets to each other with a constant speed in simulations. At first, we measure network aggregate throughput. The servers were set to send packets with a rate of 100 Mbps in a period of simulation. Scenarios 1, 2, and 3 ran three times for over 30 min each time. We measure the average traffic that is received on each server and accumulate them together as the network aggregate throughput. Table IV shows the aggregate throughput as the percentage of the maximum theoretical throughput, that is, the ideal bisection bandwidth.

Next, we measured network ETE communication delay. The servers were set to send packets with a rate from 100 Mbps down to 30 Mbps. Scenarios 1 and 3 ran 5 min under different traffic. We measure the average ETE delays and plot the results in the Figure 12.

Table IV. Aggregate throughput.

Network	Scenario 1	Scenario 2	Scenario 3
MatrixDCN	64.7%	92.3%	96.8%
Fat-tree	62.1%	93.5%	97.6%

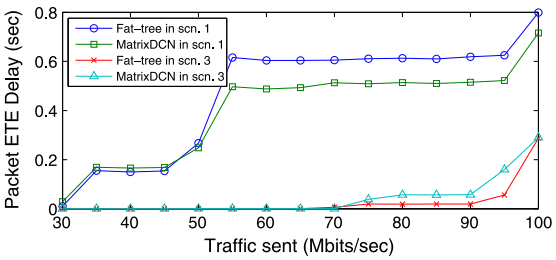


Figure 12. ETE delay in fat-tree and MatrixDCN.

Table IV indicates that the network throughput of MatrixDCN is almost the same as that of fat-tree, and the result of the simulation is close to the theoretical value analyzed in the previous section. Figure 12 shows that the

ETE communication delay in MatrixDCN is close to fat-tree too. The network performance in scenario 1 is worse than in scenario 3 on both the throughput and ETE delay, because load imbalance occurred in scenario 1, which caused too much traffic, was forwarded to the same link, and resulted in network congestion. In scenarios 2 and 3, a group of low-speed links are replaced by one high-speed link, so the problem of network load imbalance is greatly relieved.

8. CONCLUSIONS

In this paper, we present a novel network architecture MatrixDCN for large-scale cloud data centers. Our simulation shows that its performance is close to fat-tree. Additionally, MatrixDCN has some salient advantages: (1) its structure is simple and clear, which is helpful for reducing the cost of network management and maintenance, and (2) it supports extra-large-scale networks. A 2D MatrixDCN network can accommodate 4/3 to two times more servers than fat-tree; (3) it has good scalability. It is easy to expand or to shrink the scale of a MatrixDCN network; and (4) its routing algorithm FAR is simple and effective with only dozens of route entries.

Moreover, a new concept of the NRT is imported in FAR. FAR computes the paths to be avoided according to the failed links and removes these paths from possible routing paths, which is simple and efficient in a network with a regular topology. We think this method can also be applied to other network architectures such as fat-tree, and it would improve their routing efficiency.

In future work, a flow scheduler will be introduced to MatrixDCN because our simulations show that load imbalances seriously affect the performance of MatrixDCN. We will also support virtual machine (VM) migration in the entire network because it is a very important feature for DCN.

ACKNOWLEDGEMENTS

The authors would like to extend their sincere appreciation to the Deanship of Scientific Research at King Saud University for its funding of this research through the Research Group Project no. RGP-VPP-281, ZTE-BJTU Collaborative Research Program (No. K11L00190), and the Chinese Fundamental Research Funds for the Central Universities (No. K12JB00060).

REFERENCES

1. Dinh HT, Lee C, Niyato D, Wang P. A survey of mobile cloud computing: architecture, applications, and approaches. *Wireless communications and mobile computing* 2013; **13**(18): 1587–1611.
2. Lin K, Chen M. Energy equilibrium based on corona structure for wireless sensor networks. *Wireless communications and mobile computing* 2012; **12** (13): 1203–1214.
3. Chen M, Leung V, Mao S, Kwon T. A survey of mobile cloud computing: architecture, applications, and approaches. *Wireless communications and mobile computing* 2009; **9**(3): 405–416.
4. Greenberg A, Hamilton JR, Jain N, Kandula S, Kim C, Lahiri P, Maltz DA, Patel P, Sengupta S. Vl2: a scalable and flexible data center network. In *ACM SIGCOMM'09, ACM*, Barcelona, Spain, 2009; 51–62.
5. Greenberg A, Hamilton J, Maltz DA, Patel P. The cost of a cloud: research problems in data center networks. In *ACM SIGCOMM'08, ACM*, Seattle, WA, USA, Aug 2008; 68–73.
6. Chen M, Wen Y, Jin H, Leung V. Enabling technologies for future data center networking: a primer. *IEEE Network* July 2013; **27**(4): 8–15.
7. Al-Fares M, Loukissas A, Vahdat A. A scalable, commodity data center network architecture. In *ACM SIGCOMM'08, ACM*, Seattle, WA, USA, Aug 2008; 63–74.
8. Guo C, Guohan L, Li D, Haitao W, Zhang X, Shi Y, Tian C, Zhang Y, Songwu L. Bcube: a high performance, server-centric network architecture for modular data centers. In *ACM SIGCOMM'09, ACM*, Barcelona, Spain, Aug 2009; 63–74.
9. Moy JT. OSPF: anatomy of an Internet routing protocol. *Addison-Wesley Professional* 1998.
10. Li D, Mingwei X, Zhao H, Xiaoming F. Building mega data center from heterogeneous containers. In *19th IEEE International Conference on Network Protocols (ICNP), IEEE*, Vancouver, BC Canada, Oct 2011; 256–265.
11. Guo C, Haitao W, Tan K, Shi L, Zhang Y, Songwu L. Dcell: a scalable and fault-tolerant network structure for data centers. *ACM SIGCOMM Computer Communication Review, ACM* Aug 2008; **38**(4): 75–86.
12. Li D, Guo C, Haitao W, Tan K, Zhang Y, Ficonn SL. Using backup port for server interconnection in data centers. In *IEEE INFOCOM'09, IEEE*, Rio de Janeiro, Brazil, 2009; 2276–2285.
13. Liao Y, Yin D, Gao L. Dpillar: scalable dual-port server interconnection for data center networks. In *19th International Conference on Computer Communications and Networks (ICCCN), IEEE*, Zurich, Switzerland, 2010; 1–6.
14. Guo D, Chen T, Li D, Liu Y, Liu X, Chen G. BCN: expansible network structures for data centers using hierarchical compound graphs. In *IEEE INFOCOM'11, IEEE*, Shanghai, China, 2011; 61–65.
15. Gyarmati L, Anh Trinh T. Scafida: a scale-free network inspired data center architecture. In *ACM SIGCOMM'10, ACM*, New Delhi, India, Aug 2010; 4–12.

16. Curtis AR, Carpenter T, Elsheikh M, López-Ortiz A, Keshav S. Rewire: an optimization-based framework for unstructured data center network design. In *IEEE INFOCOM'12, IEEE*, Orlando, Florida USA, 2012; 1116–1124.
17. Farrington N, Porter G, Radhakrishnan S, Bazzaz HH, Subramanya V, Fainman Y, Papen G, Vahdat A. Helios: a hybrid electrical/optical switch architecture for modular data centers. In *ACM SIGCOMM'11, ACM*, Toronto, ON, Canada, Aug 2011; 339–350.
18. Wang G, Andersen DG, Kaminsky M, Papagiannaki K, Ng TS, Kozuch M, Ryan M. c-through: part-time optics in data centers. *ACM SIGCOMM Computer Communication Review, ACM* Aug 2010; **41**(4): 327–338.
19. Halperin D, Kandula S, Padhye J, Bahl P, Wetherall D. Augmenting data center networks with multi-gigabit wireless links. In *ACM SIGCOMM'11, ACM*, Toronto, ON, Canada, Aug 2011; 38–49.
20. Zhou X, Zhang Z, Zhu Y, Li Y, Kumar S, Vahdat A, Zhao BY, Zheng H. Mirror mirror on the ceiling: flexible wireless links for data centers. *ACM SIGCOMM Computer Communication Review* 2012; **42** (4): 443–454.
21. Touch J, Perlman R. *Transparent interconnection of lots of links (trill): problem and applicability statement[online]*, 2009. Available online from: http://tools.ietf.org/html/rfc5556 [accessed on July 1, 2014].
22. *Cisco. Fabricpath [online]*. Available online from: http://www.cisco.com/en/US/netsol/ns1151/index.html [accessed on July 1, 2014].
23. Kim C, Caesar M, Rexford J. Floodless in seattle: a scalable ethernet architecture for large enterprises. In *ACM SIGCOMM'08, ACM*, Seattle, WA, USA, Aug 2008; 3–14.
24. Mudigonda J, Yalagandula P, Al-Fares M, Mogul JC. Spain: cots data-center Ethernet for multipathing over arbitrary topologies. In *Proceedings of the 7th USENIX conference on Networked systems design and implementation, USENIX Association*, San Jose, CA, USA, Mar 2010; 18–18.
25. Mudigonda J, Yalagandula P, Mogul J, Stiekes B, Pouffary Y. NetLord: a scalable multi-tenant network architecture for virtualized data centers. In *ACM SIGCOMM'11, ACM*, Toronto, ON, Canada, Aug 2011; 62–73.
26. Katz D, Ward D. *Bidirectional forwarding detection(bfd) for ipv4 and ipv6 (single hop))[online]*, 2010. Available online from: http://tools.ietf.org/html/rfc5881 [accessed on July 1, 2014].
27. Sem-Jacobsen FO, Skeie T, Lysne O, Duato J. Dynamic fault tolerance in fat-trees. *IEEE Transactions on Computers* April 2011; **60**(4): 508–525.
28. Mysore RN, Pamboris A, Farrington N, Huang N, Miri P, Radhakrishnan S, Subramanya V, Vahdat A. Portland: a scalable fault-tolerant layer 2 data center network fabric. In *ACM SIGCOMM'09, ACM*, Barcelona, Spain, Aug 2009; 39–50.
29. Walraed-Sullivan M, Mysore RN, Tewari M, Zhang Y, Marzullo K, Vahdat A. Alias: scalable, decentralized label assignment for data centers. In *Proceedings of the 2nd ACM Symposium on Cloud Computing, ACM*, Cascais, Portugal, 2011.

AUTHORS' BIOGRAPHIES

Yantao Sun is a lecturer in the School of Computer and Information Technology, Beijing Jiaotong University, Beijing, China. He received his PhD degree from the Institute of Software Chinese Academy of Sciences in 2006. His research interests include cloud computing, data center network, wireless sensor network, the Internet of Things, multimedia communication, and network management. He published more than 20 papers on international conferences and journals and applied tens of invention patents.

Min Chen (minchen@ieee.org) is a professor in School of Computer Science and Technology at Huazhong University of Science and Technology. He is the Chair of IEEE Computer Society (CS) Special Technical Communities on Big Data. He was an assistant professor in School of Computer Science and Engineering at Seoul National University (SNU) from September 2009 to February 2012. He was R&D director at Confederal Network Inc. from 2008 to 2009. He worked as a post-doctoral fellow in Department of Electrical and Computer Engineering at University of British Columbia (UBC) for three years. Before joining UBC, he was a post-doctoral fellow at SNU for one and half years. He received the Best Paper Award from IEEE ICC 2012, and Best Paper Runner-up Award from QShine 2008. He has more than 180 paper publications, including 85 SCI papers. He serves as editor or associate editor for Information Sciences, Wireless Communications and Mobile Computing, IET Communications, IET Networks, Wiley I. J. of Security and Communication Networks, Journal of Internet Technology, KSII Trans. Internet and Information Systems, International Journal of Sensor Networks. He is managing editor for IJAACS and IJART. He is a guest editor for IEEE Network, IEEE Wireless Communications Magazine, and so on. He is the co-chair of IEEE ICC 2012-Communications Theory Symposium, and the co-chair of

IEEE ICC 2013-Wireless Networks Symposium. He is general co-chair for IEEE CIT-2012 and Mobimedia 2015. He is the general vice chair for Tridentcom 2014. He is the keynote speaker for CyberC 2012 and Mobiquitous 2012. He is a TPC member for IEEE INFOCOM 2013 and INFOCOM 2014. His research focuses on Internet of Things, Big Data, machine to machine communications, Body Area Networks, E-healthcare, mobile cloud computing, ad hoc cloudlet, Cloud-Assisted Mobile Computing, ubiquitous network and services, and multimedia transmission over wireless network, and so on.

Limei Peng (auroraplm@ajou.ac.kr) is an assistant professor with the department of Industrial Engineering, Ajou University, South Korea. Before she joined Ajou University, she worked as an associate professor in School of Electronic and Information Engineering, Soochow University, China, for more than 2 years. Her research interests fall in Optical communications, Cloud computing, Datacenter networks, and Software defined networks.

Mohammad Mehedi Hassan an assistant professor of the Information Systems Department at the College of Computer and Information Sciences, King Saud University, Riyadh, Saudi Arabia. He received his PhD degree in Computer Engineering from Kyung Hee University, South Korea in February 2011. He has authored and co-authored more than 70 publications including refereed IEEE/ACM/Springer journals, conference papers, books, and book chapters. His research interests include cloud collaboration, media cloud, sensor-cloud, mobile Cloud, IPTV, and wirless sensor network.

Abdulhameed Alelaiwi is an assistant professor at Department of Software Engineering, King Saud University, Riyadh, Saudi Arabia.