meGautz: A High Capacity, Fault-tolerant and Traffic Isolated Modular Datacenter Network

Feng Huang, Yiming Zhang*, Dongsheng Li*, Jiaxin Li, Jie Wu, Kaijun Ren, Deke Guo, Xicheng Lu

Abstract—The modular datacenter networks (MDCN) comprise inter- and intra-container networks. Although it simplifies the construction and maintenance of mega-datacenters, interconnecting hundreds of containers and supporting online data-intensive services is still challenging. In this paper, we present meGautz, which is the first inter-container network that isolates inter- and intra-container traffic, and it has the following advantages. First, meGautz offers uniform high capacity among servers in the different containers, and balances loads at the container, switch, and server levels. Second, it achieves traffic isolation and allocates bandwidth evenly. Therefore, even under an all-to-all traffic pattern, the inter- and intra-container networks can deal with their own flows without interfering with each other, and both can gain high throughput. meGautz hence improves the performance of both the entire MDCN and individual servers, for there is no performance loss caused by resource competition. Third, meGautz is the first to achieve as graceful performance degradation as computation and storage do. Results from theoretical analysis and experiments demonstrate that meGautz is a high-capacity, fault-tolerant, and traffic isolated inter-container network.

Index Terms—Modular datacenter network, inter-container network, traffic isolation, fault tolerance.

1 Introduction

Modular datacenter (MDC) uses shipping-containers as large pluggable building blocks to construct mega-datacenters, which is considered as an ideal solution for the next-generation datacenters [1], [2], [3], [4]. Modular datacenter network (MDCN) is the fundamental component of MDC, so it has increasingly attracted significant attention from cloud providers, hardware vendors, and academia [1], [2], [5]. Due to containerization, MDCN comprises the intra- and inter-container network structures. Generally, the intra-container network organizes 1,200-2,500 servers in each standard 20- or 40-foot container; meanwhile the inter-container network connects 400-800 containers of this kind to build large MDC [6], [7], [8], [9], [10], [11].

MDC is the underlying infrastructure of modern Online Data Intensive services (OLDI) [12], [13], such as web search, social networking and scientific computing. The growing demand of these OLDI services have driven MDCN to an enormous scale, even accommodating thousands of containers. So for the inter-container networks, there are three requirements to be met.

High inter-container bandwidth: OLDI services deployed in MDC can run at a massive scale, where each operation needs to process data spanning tens of thousands of servers over tens to hundreds of containers. With high bandwidth between containers and high throughput of MDCN, every worker node is able to send results to the aggregation node to assemble the final result in a timely manner. The fewer the number of worker nodes that miss their deadline, the better the quality of user experience and revenue.

- *F. Huang, Y. Zhang, D. Li, J. Li, K. Ren, D. Guo and X. Lu are with National University of Defense Technology, Changsha, CN, 410073.*
  **Yiming Zhang and Dongsheng Li are the corresponding authors.*
  E-mail: {ymzhang, dsli}@nudt.edu.cn
- *J. Wu is with Department of Computer and Information Sciences, Temple University, Philadelphia, USA, 19122.*

Manuscript received October 15, 2015; revised March 3, 2016; accepted March 20, 2016.

Smooth performance decrease: The current MDC are built of inexpensive commodity hardware; the failure of devices has become a norm. Since the devices in containers are prefabricated, their failure is difficult to repair, due to operational and space constraints. Therefore, inter-container networks should be able to tolerate the continuous hardware failures gracefully, making network capacity decrease as smoothly as computation and storage capacity do.

No resource competition between inter- and intra-container flows: Every user's request to an OLDI service can generate tens of thousands of flows from worker nodes to aggregation nodes [12], [13], including the inter- and intra-container flows. Since the inter-container flows have to traverse multiple containers to reach the destinations, if they use one or several same links as the intra-container flows do, resource competition will occur. As MDC scale out, the inter-container flows increase drastically and preempt the available resource of the intermediate containers greedily and rapidly. While the intra-container flows would have their own bandwidth drop sharply and may even starve, leading to the performance decline of the entire MDCN. Hence, the problem of resource competition is nontrivial and it must be solved by isolating the intra- and inter-container traffic.

However, the current inter-container networks do not achieve all of the above goals simultaneously, such as Helios [14], uFix [15], and MDCube [9]. First, their network performance drops too fast in the presence of failure. For example, MDCube behaves best on fault-tolerance in the existing proposals. But when 10% of the servers or switches fail, its throughput decreases by up to 25.5% and 60.9%, respectively. Second, none of the proposals addresses the problem of resource competition. Also taking MDCube as an example, when its servers have an all-to-all traffic from two to ten containers, its inter-container throughput increases by 56.4%, whereas the intra-container throughput is reduced by 82.5%, resulting in an obvious decrease of the per-server throughput by 16.2% [9]. If all the servers in MDCube generate all-to-all traffic, its inter-container flows nearly exhaust all the bandwidth and starves the intra-container flows. Thus, the increase in number of failures (that cannot be serviced) and resource competition are the main reasons of inter-container networks, which affect OLDI applications severely.

In this paper, we propose meGautz[1], a high-capacity, fault-tolerant, and traffic isolated inter-container network. meGautz interconnects the containers effectively, and it is able to offer higher capacity and stronger fault-tolerance than other approaches. Besides, meGautz achieves traffic isolation between the inter- and intra-container networks. Hence, it eliminates the resource competition and allocates bandwidth evenly to inter- and intra-container flows. meGautz is a server-centric structure, since it implements routing intelligence on servers and relies on servers to compute routing paths.

meGautz chooses the SCautz-containers [16], whose inner network is SCautz, to construct MDC. Generally, containers are preprovisioned with spare devices to realize "service-free" [1]. By using these resources, SCautz builds a hierarchical structure and achieves many nice characteristics for meGautz to isolate traffic and tolerate faults. In meGautz, each container is regarded as a virtual node, and these virtual nodes are connected to form an optimized topology with a constant degree. Based on it, traffic-isolated and load-balancing routing algorithms are designed. The traffic-isolated routing multiplexes the spare resources to forward packets through intermediate containers without affecting the inner ones, so as to eliminate resource competition. The load-balancing routing distributes traffic evenly at the container, switch, and server levels, improving the network capacity and resource utilization. Furthermore, the multiple paths among containers, especially the neighboring containers, realizes smooth performance degradation of meGautz and the entire MDCN.

To the best of our knowledge, meGautz is the first inter-container network that takes traffic isolation as a design goal (and realizes it). The main contributions are as follows.

1. meGautz provides high and uniform server-to-server network capacity across the different containers. In meGautz, the reserved high-speed ports of multiple commodity switches in SCautz are trunked as virtual ports, and then are linked by the optical fibers to organize containers into a Kautz [17] topology. Both the construction method and topology of meGautz guarantee the high network capacity together with its novel routing algorithms. Moreover, the cost of meGautz is low, for there is no extra devices are introduced but optical fibers.
2. meGautz is the first to achieve performance degradation as gracefully as computation and storage capacity do. High path redundancy between containers realizes better resilience to failure.
3. meGautz isolates its traffic from the intra-container networks. By utilizing the spare links, meGautz is able to forward the traffic across the containers without interfering with theirs. For the all-to-all traffic, experimental results show that the intra-container throughput of each container in meGautz can be achieved and maintained as high as the ABT (Aggregation Bottleneck Throughput) of an individual SCautz-container; meanwhile, the inter-container throughput increases linearly as containers rise for no resource competition. Therefore, the bandwidth is allocated to inter- and intra-container networks evenly, and higher network capacity can be obtained without performance loss.

The remainder of the paper is organized as follows. Section II discusses the nature of MDC and OLDI, and analyzes the design goals and challenges of MDCN. Section III describes meGautz's architecture. Section IV proposes the routing algorithms. Section V conducts experiments to evaluate meGautz. Section VI introduces the related works. Finally, section VII concludes the paper.

2 Preliminaries

We start by describing the modular datacenters (MDC) and Online Data Intensive (OLDI) services. We then introduce Kautz graph and SCautz structure. At last, we analyze design goals that inter-container networks should realize.

2.1 Modular Datacenter and Online Data Intensive Services

MDC is constructed of 20- or 40-feet standard containers. The main procedure is that intra-container networks first get thousands of servers linked, which have been set up and packaged in each container; then, an inter-container network connects these containers together. Once hooked up to power, a cooling infrastructure, and the Internet, MDC can provide services at any location in the world. Besides high mobility, deployment flexility, and lower ownership and management cost, the most outstanding feature of MDC is that containers run in a particular "service-free" manner. That is, containers as a whole are never repaired during their lifetime (e.g., 3- 5 years), as long as the overall performance meets an engineered minimum criterion [1], [2]. Compared to traditional datacenter networks, this feature poses new challenges on designs of inter-container networks. To realize "service-free", the extra resources are deployed into containers for fault-tolerance. But they have never been considered by the current approaches.

OLDI applications need to access big data sets simultaneously, which might be distributed over thousands of servers in different containers. The processing of OLDI applications includes two phases: *partition and aggregate* (or *map and reduce*). In the first phase, applications *partition* each request, and assign them to workers to process their local data, respectively. In the second phase, every aggregator fetchs intermediate results from all other workers to *aggregate* the final response, leading to a typical all-to-all data shuffle pattern. If these all-to-all traffic across multiple containers can be effectively supported, data shuffles will finish fast without missing deadlines.

2.2 Kautz and SCautz

A directed Kautz graph [17] with degree d and diameter k, denoted as $K(d,k)$, is a digraph with $d^k + d^{k-1}$ nodes and $d^k + d^{k+1}$ edges. Let $Z_d = \{0, 1, \cdots d\}$ be an alphabet of $d+1$ letters. We define Kautz strings as $Z_d^k = \{x_1 \cdots x_k | x_i \in Z_d,\ x_i \neq x_{i+1}\ and\ 1 \leq i < k\}$, and their consecutive letters are different. So the nodes in Kautz are represented by Kautz strings. There is an edge from node X to Y, if Y is a left-shifted version of X, that is, if $X = x_1 \cdots x_k$, $Y = x_2 \cdots x_k x_{k+1}, x_{k+1} \in Z_d,\ and\ x_{k+1} \neq x_k$, then an edge $X \rightarrow Y$ exists. The undirected Kautz structure we use, denoted as $UK(d,k)$, is obtained by omitting the directions of the edges and keeping the loops. Assumes there is a loop between X and Y, comprising edges from X to Y and Y to X. X is also a left-shifted version of Y, and $X = x_3 \cdots x_{k+1} x_{k+2}$. So, $x_1 = x_3 = \cdots = x_{k+2}, x_2 = x_4 = \cdots = x_{k+1}$.

1. meGautz represents "meGa datacenter network based on Kautz-graphs.

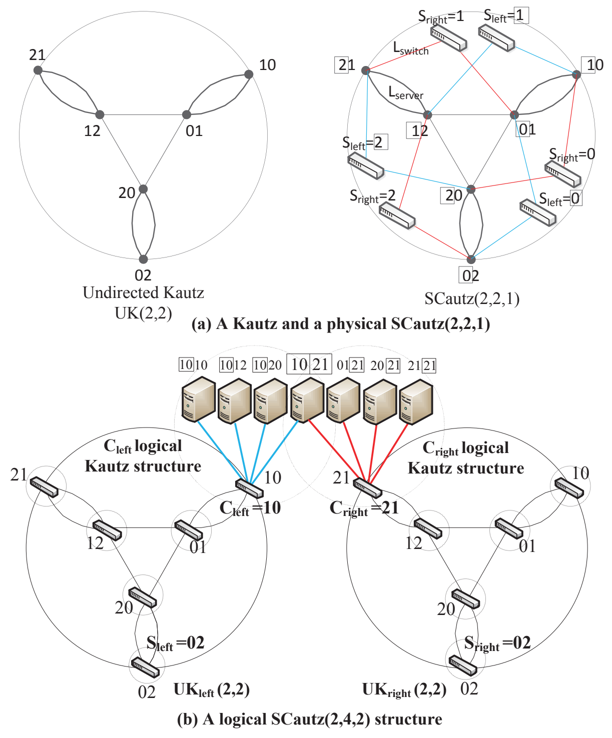


Fig. 1: Kautz graph and SCautz structure.

It has loops only between the $d^2 + d$ nodes of the form $(abab\cdots)$, e.g., (01, 10) and (02, 20), as shown in Figure 1 (a). So it is regular, which is not like the general undirected Kautz graph. Note that, in the rest of this paper, the term "Kautz" is used to denote this kind of undirected Kautz graph. meGautz and SCautz are both designed based on Kautz graph.

meGautz chooses SCautz [16] as the intra-container network, because it behaves much better on high-throughput and fault-tolerance than other proposals. Besides, SCautz is able to realize intra-container routing all by the links between servers, reserving the over-provisioned switches and links between servers and switches for meGautz to design inter-container routing algorithm and achieve traffic isolation.

At first, SCautz interconnects the NIC ports of servers directly, forming a physical Kautz topology with degree d and diameter k, i.e, $UK(d,k)$, as its base structure. And then, it over-provisions tens of commodity switches to process bursts of traffic and tolerate faults. Therefore, all the servers and switches constitute a complete structure of SCautz as follows, denoted as $SCautz(d,k,t)$, where t represents the identifier length of switches.

In SCautz, the definition and identifiers of servers are identical to those of the nodes in Kautz graph. The spare switches are divided into two categories, referred to S_{left} and S_{right}, respectively. S_{left} (resp. S_{right}) connects the servers whose identifiers' t leftmost (rightmost) letters are identical. We let these t leftmost letters (resp. rightmost letters) be S_{left}'s (resp. S_{right}'s) identifiers. Thereby each switch and its $n = d^{k-t}$ servers form a "cluster", denoted as C_{left} (resp. C_{right}). In this paper, we will not distinguish S and C, i.e., the identifiers of clusters and switches are the same. Furthermore, if clusters are treated as virtual nodes, it has been proven that all C_{left} and C_{right} form two higher-level logical Kautz structures with diameter t, respectively, denoted as $UK_{left}(d,t)$ and $UK_{right}(d,t)$ [16]. So in SCautz, every server connects with two switches, and belongs to two clusters of two types at the same time. Accordingly, there are two types of links in SCautz: L_{server} and L_{switch}. L_{server} connect servers directly to form SCautz's base structure, which can provide a throughput as high as BCube [16]. Meanwhile, L_{switch} connects the servers with spare switches, which are reserved and can be used by meGautz to isolate the inter- and intra-container traffic.

Figure 1 (a) shows a physical structure of $SCautz(2,2,1)$. Its base structure $UK(2,2)$ is built by connecting 6 servers directly via links L_{server}. Based on $UK(2,2)$, switches S_{left} and S_{right} connect the corresponding servers via links L_{switch}, to form the complete structure of $SCautz(2,2,1)$. Figure 1 (b) illustrates a logical structure of $SCautz(2,4,2)$. We can see that the switch $S_{left} = 10$ connects servers 1010, 1012, 1020, 1021, whose 2 leftmost letters are 10, forming a cluster $C_{left} = 10$. All the C_{left} build a logical Kautz structure, i.e., $UK_{left}(2,2)$. Analogously, $S_{right} = 21$ connects servers 1021, 0121, 2021, 2121, whose 2 rightmost letters are 21, forming a cluster $C_{right} = 21$. All the C_{right} build a $UK_{right}(2,2)$. Note that the server 1021 is the member of cluster $C_{left} = 10$ and $C_{right} = 21$, simultaneously.

2.3 Design goals and challenges

Connecting SCaut-containers to build an efficient MDC and improve the quality of OLDI services, meGautz should address several challenges and achieve the following design goals.

High inter-container capacity can accelerate all-to-all data shuffles of OLDI applications. Generally, the inter-container network improves communication capacity by over-provisioning a large number of parallel links with high bandwidth between containers, and organizing them into an excellent topology with desired network properties.

Typically, there are two representative solutions for container interconnection: introducing new high-end switches [14], [18] or connecting existed components [9], [15]. The former incurs either high cost or over-subscription, or even both; while the latter builds a non-blocking network at a low cost. So meGautz chooses the latter approach, and it interconnects the high-speed ports (e.g., 10- or 40 Gbps) of switches in containers via optical fibers. Note that these links can never be the bottlenecks. For a network, node degree and network diameter are two critical metrics to be determined with caution. We design meGautz base on Kautz graph, for it achieves a near-optimal tradeoff between them, and has better bisection width and bottleneck degree. In meGautz, high node degree means more neighboring containers. But its total amount of switch ports in each container is fixed, so the ports and links connected to each neighbor will be less, and the bandwidth is limited. We consider that meGautz's degree should be low and the bandwidth between neighboring containers should be as high as possible. Firstly, to improve performance, the nearby resources in the neighboring containers are always scheduled to deploy and run OLDI services or virtual machines [19]. Secondly, the intra- and inter-container networks should both be non-blocking, and links between containers are never the bottlenecks. Therefore, we construct meGatuz based on an undirected Kautz with a constant degree of 2, whose degree is lowest. The tradeoff we pay is the slightly reduced all-to-all throughput of the whole MDCN.

Fault-tolerance helps the network retain its merits and capacity, when failures happen and grow in "service-free" MDC. In MDCN, component failures, such as links, servers, and switches, could affect both of intra- and inter-container networks. For a

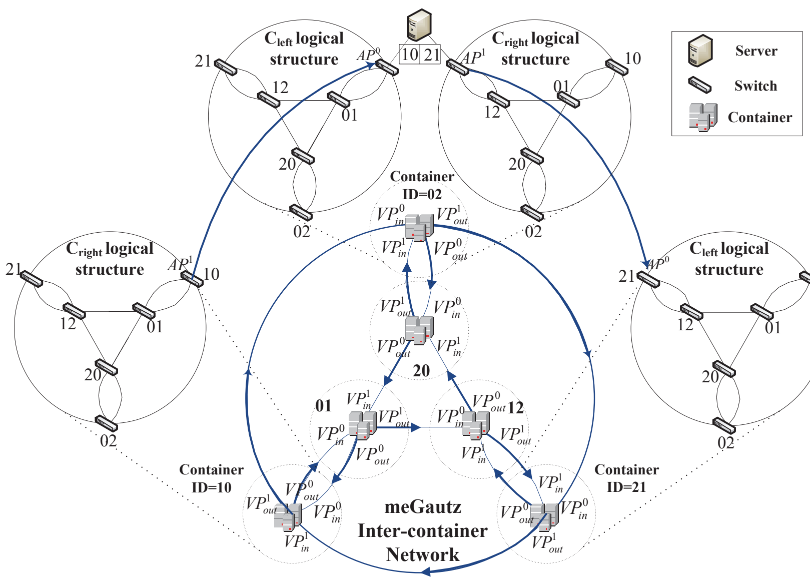


Fig. 2: A meGautz interconnecting six $SCautz(2,4,2)$ containers.

server-centric MDCN, server failure degrades the performance of MDC's computation, storage, and network simultaneously. Intuitively, network capacity should not decrease faster than computation and storage. Otherwise, it will become the fatal cause that hurts OLDI's quality significantly.

As far as we know, none of the inter-container networks achieves the above fault-tolerance goal. There are two problems that must be solved: (i) single-point failures of switches between containers, e.g., when an inter-container switch in MDCube [9] fails, all the traffic through that switch has to be re-routed via another server in the same container, which brings significant overhead to that server. (ii) the inefficient fault-tolerant path, whose length is much longer [9]. Hence, meGautz should not only deploy a rich connection between the neighboring containers, but also provide multiple parallel paths for any pair of them, so that it can construct a fault-tolerant path to bypass the failed components at a low cost.

Traffic isolation is able to avoid resource competition between the inter- and intra-container networks. For MDCN, which is built by connecting containers' existing components, it is common to rely on the internal resources to forward the inter-container flows. However, today's OLDI and other applications run on the same MDC, leading to burst inter- and intra-container flows under soft-real-time constraints. If these two types of flows keep interfering with each other, lots of flows will have their deadlines expire.

The more internal resources that inter-container flows occupy, the more severe resource competition will become. To get rid of this problem, the following challenges of three aspects must be addressed. Firstly, for the server-centric MDCN, each server should connect with at least one ingress and egress port. So it can receive inter-container packets directly, and route to the next container immediately. Secondly, inter- and intra-container routing must be decoupled from each other completely. Being multiplexed or extended to support inter-container routing, the intra-container routing algorithms will be used to transfer both inter- and intra-container flows, hence incurring resource competition. At last, inter-container routing should make full use of the spare resources without interfering intra-container routing. So, the inter- and intra-container traffic could be isolated to eliminate the resource competition.

Although SCautz-containers have many great network properties, and over-provision switches and links. It is still challenging for meGautz to interconnect hundreds of them to build massive MDC, having the above goals to be achieved.

3 MEGAUTZ ARCHITECTURE

Similar to SCautz [16], the meGautz architecture leverages Kautz graph [17] to achieve low diameter and high bandwidth. In this section, we introduce and explain how meGautz interconnects SCautz-containers in detail.

We construct meGautz by interconnecting containers into a Kautz topology with a constant degree of 2, denoted as $UK(2,m)$. Firstly, every container needs just two "virtual ingress/egress ports" to form a $UK(2,m)$ topology. So each "virtual port" can gain as high an aggregation bandwidth as possible by trunking 1/4 number of switches' high-speed ports together. Secondly, in the $UK(2,m)$ Kautz graph, there are four edge-disjoint paths between any pair of nodes; in meGautz, this means four parallel paths from source to destination containers. There are rich connections at the link, server, and container levels. Thus, both construction methods and optimized topology of meGautz realize rich connections at the link, server and container levels, and improve the MDCN's overall performance. Moreover, the number of nodes in a $UK(2,m)$ increases relatively slowly as m increases, so meGautz can support MDC of various scales.

In meGautz, a container's identifier is denoted as MC, which can be represented by a Kautz string with base 2 and length m (i.e., Z_2^m). For the servers, links, and switches, their identifiers must add the related container's identifier as a prefix. For instance, a server N belongs to a container MC, then its identifier will be $< MC \cdot N >$.

In a SCautz, there are $(d^t + d^{t-1}) \times 2$ switches of two types. Supposing that each switch has sp ($sp \geq d^{k-t}$) low-speed (B_{low} Gbps) ports and hp high-speed (B_{high} Gbps) ports, we divide its high-speed ports equally, and trunk every $hp/2$ high-speed ports as one "aggregation port" of a switch, denoted as AP^0 and AP^1, respectively. It is easy to know that the bandwidth of AP^0 and AP^1

are $B_{high} \times (hp/2)$. While at the container level, we view all AP^0 of all the switches $MC \cdot S_{right}$ in a container MC as a "virtual egress-port", denoted as VP^0_{out}, and all AP^1 of all the switches $MC \cdot S_{right}$ as the other one, denoted as VP^1_{out}. Analogously, we can obtain the two "virtual ingress-ports" of MC, denoted as VP^0_{in} and VP^1_{in} by aggregating AP^0 and AP^1 of the switches $MC \cdot S_{left}$.

Using the four "virtual ports" of each container, we construct meGautz as follows. Suppose that containers $MC = \{c_1c_2 \cdots c_m | c_i \in [0,1,2], c_i \neq c_{i+1},\ \ and\ \ 1 \leq i < m\}$, $MC_0 = \{c_0c_2 \cdots c_m | c_0 \in [0,1,2]\ and\ c_0 \neq c_2\}$, $MC_1 = \{c_2 \cdots c_m u | u \in [0,1,2]\ and\ u \neq c_m\}$, $MC_2 = \{c_2 \cdots c_m v | v \in [0,1,2]\ and\ v \neq c_m\}$. MC and MC_0 are the left-neighbor containers of MC_1 and MC_2, and MC_1 and MC_2 are the right-neighbor containers of MC and MC_0. Let $c_0 < c_1$ and $u < v$. According to the complete ordering relation of node identifiers, defined in Definition 1, $MC_0 < MC$ and $MC_1 < MC_2$ hold.

Definition 1. (**complete ordering relation**) In a Kautz $UK(d,k)$ graph, let node $X = \{x_1 \cdots x_k | x_i \in Z_d,\ x_i \neq x_{i+1}\ and\ 1 \leq i < k\}$ and $Y = \{y_1 \cdots y_k | y_i \in Z_d,\ y_i \neq y_{i+1}\ and\ 1 \leq i < k\}$, then $X > Y$ holds, if $x_1 > y_1$ or $x_i > y_i$ if $x_1 = y_1, \cdots, x_{i-1} = y_{i-1}, 1 < i \leq k$.

Viewed as "virtual nodes", each container in meGautz has two right and left neighbors, respectively. So, for its VP_{out}, we connect the VP^0_{out} to the right-neighboring container with little identifier, and the VP^1_{out} to the right-neighboring one with big identifier. Meanwhile for its VP_{in}, we connect the VP^0_{in} to the left-neighboring container with little identifier, and VP^1_{in} to the left-neighboring one with big identifier. The "virtual links" between "virtual ports", which consist of optical fibers between switches, are denoted as VL. Thus, VP^0_{out} and VP^1_{out} of MC connect with the VP^1_{in} of MC_1 and MC_2; VP^0_{out} and VP^1_{out} of MC_0 connect with the VP^0_{in} of MC_1 and MC_2, respectively. Until all the containers get linked, we obtain meGautz's $UK(2,m)$ Kautz topology logically.

Physically, the optical fibers connect high-speed ports of switches that get containers wired in meGautz. For simplicity, we focus on the interconnection of switches' trunked ports (i.e., AP), and denote the "links" connecting APs as $L_{container}$. As mentioned, containers MC and MC_1 are neighbors, so we define the switch $MC \cdot S_{right}$ and $MC_1 \cdot S_{left}$ as "peer switches", if $S_{right} = S_{left}$. Each VP is a group of AP. If MC wants its VP_{out} to be connected with MC_1's VP_{in}, then all pairs of peer switches in them must get the corresponding APs wired. Therefore, the AP^0 (resp. AP^1) of $MC \cdot S_{right}$ connects with the AP^1 of $MC_1 \cdot S_{left}$ (resp. $MC_2 \cdot S_{left}$). In this way, container MC and MC_0 get wired, and so do all other containers and the whole meGautz. Between the neighboring containers, meGautz can provide $d^t + d^{t-1}$ switch-disjoint parallel paths and an inter-container bandwidth of up to $(d^t + d^{t-1}) \times B_{high} \times (hp/2)$ Gbps.

Figure 2 illustrates a meGautz with a $UK(2,2)$ Kautz topology, which connects six $SCautz(2,4,2)$ containers. Taking containers 02, 12 and their two right neighbors 20, 21 as examples, since $02 < 12$ and $20 < 21$, container 02 connects its VP^0_{out} with 20's VP^0_{in}, and VP^1_{out} with 21's VP^0_{in}; Meanwhile, container 12 connects its VP^0_{out} with 20's VP^1_{in}, and VP^1_{out} with 21's VP^1_{in}. For switches, since the switch $S_{right} = 21$ in container 02 and $S_{left} = 21$ in container 21 are peer switches, there exist fiber cables connecting the aggregation port AP^1 and AP^0, respectively. Therefore, if each switch is equipped with four 10Gbps high-speed ports, the bandwidth between two containers is 120Gbps with six switch-disjoint paths.

In addition, management complexity is an important problem we try to solve when design and construct meGautz. First, $UK(2,m)$ is the simplest Kautz structure with the smallest degree. So each container only needs to be linked to 3-4 neighbors. Second, all the switches are wired to just two "peer-switches" in two neighboring containers, respectively. And, the "peer-switches" have the same identifier suffix and easy to be recognized by workers. As for wiring, the interior size of a 40-feet container is 12*2.35*2.38m^3, so there is enough space to accommodate the wires. The inner-container wires are inside the racks and do not go out. The inter-container wires are between the SCautz units and we place them on top of the racks.

4 ROUTING IN MEGAUTZ

Routing in MDCN comprises inter- and intra-container flows, and their traffic should be isolated. In this section, we focus on the inter-container routing, and propose a suite of traffic isolated, load-balanced, and fault-tolerant routing algorithms in meGautz.

4.1 Traffic isolated Routing

According to meGautz's topology, we design a traffic-isolated inter-container routing algorithm, called meRouting, based on the following facts. First, every server connects with two switches, i.e., S_{left} and S_{right}, whose high-speed ports are the ingress and egress of a container, respectively. So each server can receive packets from two neighboring containers via S_{left} (resp. S_{right}) and forward to the other two containers via S_{right} (resp. S_{left}) directly. Second, the switches, and links L_{switch} between switches and servers are are over-provisioned in SCautz-containers to deal with traffic bursts and device failure. SCautz is able to complete the intra-container routing all by the links between servers, without using the switches and links L_{switch}. So, these spare resources can be utilized by meRouting to transfer inter-container flows without competing with the inter-container ones. The goal of traffic isolation between meGautz and SCautz is achieved.

meRouting adopts a hierarchical approach, including three main steps: 1) Compute a logical path of containers using either shortest or longest path routing algorithm, based on meGautz's $UK(2,m)$ Kautz topology [17]. 2) Choose actual servers and switches in each intermediate containers to build a physical path, entering the destination container; 3) Route to the destination server in the last-hop container.

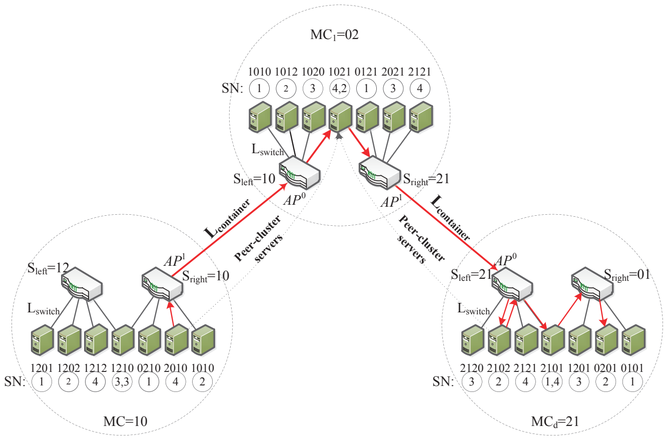


Fig. 3: Traffic isolated routing from container 02 to 21

For meRouting, step 2 builds inter-container paths from one container to the next hop. Given MC and MC_1, a server $MC \cdot X$ sends packets into MC_1 via a pair of peer-switches $MC \cdot S_{right}$ and $MC_1 \cdot S_{left}$. There are d^{k-t} servers linked to $MC_1 \cdot S_{left}$, meRouting must designate one to relay packets to the destination. We define two servers as "peer-switch servers", if they connect to a pair of peer switches. So any server has d^{k-t} peer-switch servers in each neighboring container. In SCautz, every d^{k-t} servers connecting to the same switch are grouped into a cluster. meRouting sorts and numbers them in ascending order, according to Definition 1. The sequence number is denoted as SN, and $SN \in [1, d^{k-t}]$). Since a server belongs to two clusters of two types, i.e., C_{left} and C_{right}, it has two respective SN. For example, the server 1021's SN is 4 in $C_{left} = 10$, and is 2 in $C_{right} = 21$. So server $MC \cdot X$ always has the one and only peer-switch server $MC_1 \cdot Z$ in MC_1, whose SN are equal. Hence, the inter-container path from MC to MC_1 is $MC \cdot X \to MC \cdot S_{right} \to MC_1 \cdot S_{left} \to MC \cdot Z$. Figure 3 shows two inter-container paths, MC to MC_1 and MC_1 to MC_d. $MC \cdot 2010$ and $MC_1 \cdot 1021$ are peer-switch servers, and their SNs are both 4. So are $MC_1 \cdot 1021$ and $MC_d \cdot 2102$, and their SNs are 2. Thus, the complete server path from MC to MC_d is $10 \cdot 0210 \to 02 \cdot 1021 \to 21 \cdot 2102$. In addition, we know that meRouting only takes one server and two switch hops to traverse an intermediate container by just L_{switch} links.

In container MC_d, the designated server $MC_d \cdot Z'$ may not be the destination $MC_d \cdot Y$. So meRouting also needs an intra-container routing operation. To accomplish the step 3, meRouting can only rely on L_{switch} links for avoiding resource competition. We design meGautz based on a specific $SCautz(d,k,t)$-container, in which $k = 2t$. So servers in $SCautz(d,2t,t)$ can be represented as $S_{left} \cdot S_{right}$, where $S_{left} = x_1 \ldots x_t$ and $S_{right} = x_{k-t+1} \ldots x_k$. By correcting t leftmost or rightmost letters every time, meRouting is able to compute a path from $MC_d \cdot Z'$ to $MC_d \cdot Y$, and it will never preempt the L_{server} links used by intra-container routing. Supposing that $MC_d \cdot Z' = 21 \cdot 2102$ and $MC_d \cdot Y = 21 \cdot 0201$, the path is $21 \cdot 2102 \to 21 \cdot 21 \to 21 \cdot 2101 \to 21 \cdot 01 \to 21 \cdot 0201$. Combining all sub-paths, meRouting obtains the complete path from server $10 \cdot 2010$ to $21 \cdot 0201$, as shown in Figure 3.

In summary, meRouting achieves traffic isolation successfully, and its diameter can be obtained as follows. Note that all proofs of theorems and pseudo-code of algorithms are presented in [20].

Theorem 1. For a meGautz with an $UK(2,m)$ Kautz topology built from $SCautz(d,2t,t)$ containers, the path length in meRouting between any two servers is, at most, $3m+6$.

In meGautz, meRouting delivers a high inter-container capacity at the cost of a longer diameter. This tradeoff is worthwhile for OLDI applications, as discussed previously. Due to traffic isolation, even if traversing multiple containers is required, meRouting is also able to deal with traffic bursts gracefully.

4.2 Data Distribution for All-to-all Traffic

OLDI applications generate a typical all-to-all data shuffle over containers, bringing intensive pressures on MDCN. Although the situation that an entire MDC is under all-to-all traffic is rare, the overall throughput to process it can directly reflect the MDCN's capacity. So we focus on analyzing meGautz's throughput under an all-to-all traffic pattern in theory here.

Due to traffic isolation, the all-to-all traffic in meGautz can be divided and distributed separately, and thereby high throughput is achieved. The intra-container flows, that each server launches with the other servers in the same container, are processed by SCautz via the links L_{server}. While the inter-container flows that each server launches with all servers in other containers, are processed by meGautz via the L_{switch} links and optical fibers. The overhead caused by resource competition is completely eliminated, and the following theorem is obtained.

Theorem 2. Consider that an MDC, with M containers, is under an all-to-all traffic pattern. With meRouting, the number of flows carried on a normal link L_{switch} is $\frac{(M-1)N(2A_m+A_s)}{4}$, and the number of flows carried on a virtual link VL between the VP of two neighboring containers is $\frac{(M-1)N^2A_m}{4}$, where N is the number of servers in a container, A_m is the average container path length, and A_s is the average intra-container path length of meRouting.

Theorem 2 indicates that the numbers of flows, carried on the normal links and virtual links, are different under an all-to-all traffic pattern; so are their requirements on bandwidth. The bottleneck of meGautz is determined by the ratio of capacity of VL over that of L_{switch} [9]. According to Theorem 2, the ratio can be estimated as $r = \frac{NA_m}{2A_m+A_s}$.

Thus, for a mega-datacenter [2], [9] constructed by meGautz, it accommodates 1,179,648 servers and 768 $SCautz(2,10,5)$ containers, organized into an logical $UK(2,9)$ Kautz topology. The $A_s = 4.2$, $A_m = 7.8$, so the ratio $r = 605.1$. It means that the required bandwidth of a VL, to process all-to-all traffic, is 605.1 Gbps. Assuming that SCautz adopts 48-port COTS switches with four 10 Gbps high-speed ports, a virtual port and link can provide a bandwidth of 960 Gbps. Since all high-speed links are symmetric and evenly used, each high-speed port should contribute a bandwidth of 6.3 Gbps on average. It is less than 10 Gbps. So the fiber cables between containers are not meGautz's bottlenecks, and the line rate of the normal links (i.e., 1 Gbps) can be achieved. If the degree of meGautz takes more than 2, the required bandwidth of inter-container links must be more than 579.6 Gbps. However, since the high-speed ports cannot be trunked, and each one is connected to a peer-switch in the different containers. So, every inter-container links can only gain an aggregate bandwidth of less than 480 Gbps, and they will become bottlenecks of the entire MDCN, reducing the overall throughput and per-server bandwidth.

The metric ABT (Aggregation Bottleneck Throughput) [8], [9] reflects DCN's capacity. ABT is defined as the throughput of bottleneck flow times the total number of flows, so distribution time for all-to-all traffic is the total data divided by ABT. Besides this, we introduce the inter- and intra-container throughput (ITC and IAC) to evaluate the performance of inter- and intra-container networks on processing their respective flows. For meGautz, its ITC is constrained by the L_{switch} links, and $ITC = \frac{4NM}{(2A_m+A_s)}$; meanwhile, the throughput of each container is $\frac{4N}{A_{SCautz}}$ for no influence by the inter-container traffic [16], where A_{SCautz} is the average path length in SCautz, so $IAC = \frac{4NM}{A_{SCautz}}$. Thus, ABT of the whole MDCN is a sum of ITC and IAC, and $ABT = \frac{4NM}{(2A_m+A_s)} + \frac{4NM}{A_{SCautz}}$.

4.3 Optimization and Load-balancing

meRouting distributes all-to-all traffic evenly but inefficiently. At container level, meRouting always selects the same logical container-paths and forwards traffic via the fixed virtual ports. While at server and switch levels, it always designates a fixed pair of peer-switch servers with the same SN, routing inter-container packets via two particular peer switches. Inefficient bandwidth usage may lead to a low utilization ratio and poor fault tolerance of the entire MDC. So we optimize meRouting and design a load-balancing routing algorithm, called meLBRouting, to solve this problem.

Randomizing is effective in coping with volatility of traffic in DCNs. Based on this principle, VL2 realizes an even traffic distribution among core switches [21], and so does MDCube among containers [9]. The basic idea of meLBRouting stems from VL2 and MDCube. Further, meGautz achieves load-balancing at the container, switch, and server levels.

Algorithm 1 The step2 of meLBRouting algorithm

```
Require: server $X = x_1x_2\ldots x_k$, container $MC$, egress-switch $S_{right}^{MC}(X)$, container $MC_1$, $S_{right}^{MC}(X)$ and $S_{left}^{MC_1}(X)$ are peer-switch, server $X$ and $PeerClsuterServer(X))$ in $MC_1$ are peer-switch servers,
    $X$ and $Peer_{left}(X)$ in $C_{left}(X)$ are peer servers.
Ensure: $NeighborContainerPath_{server}$
1: $NeighborContainerPath_{server} = \{X,\ \}$;
2: $Path_{server}(X, MC_1) = (X \to S_{right}^{MC}(X)) + (S_{right}^{MC}(X) \to S_{left}^{MC_1}(X)) + (S_{left}^{MC_1}(X) \to PeerClsuterServer(X))$;
3: let $Bandwidth(X \to S_{right}^{MC}(X))$ be the available bandwidth of sub-path $X \to S_{right}^{MC}(X)$ ;
4: let $Bandwidth(S_{right}^{MC}(X) \to S_{left}^{MC_1}(X))$ be the available bandwidth of sub-path $S_{right}^{MC}(X) \to S_{left}^{MC_1}(X)$;
5: let $Bandwidth(S_{left}^{MC_1}(X) \to PeerClsuterServer(X)) = max(L_{switch}^{MC_1}(S_{left}^{MC_1}(X), PeerClsuterServer(X)))$
   // the available bandwidth of sub-path $S_{left}^{MC_1}(X) \to PeerClsuterServer(X)$ is determined by max bandwidth of $d^{k-t}$ links $L_{switch}^{MC_1}(S_{left}^{MC_1}(X), PeerClsuterServer(X))$;
6: $Bandwidth(Path_{server}(X, MC_1)) = min(Bandwidth(X \to S_{right}^{MC}(X)), Bandwidth(S_{right}^{MC}(X) \to S_{left}^{MC_1}(X)), Bandwidth(S_{left}^{MC_1}(X) \to PeerClsuterServer(X)))$.
7: compute $d^{k-t} - 1$ $Bandwidth(Path_{server}(Peer_{left}(X), MC_1))$
8: find the $NeighborContainerPath_{server} = Path_{server}(Peer_{left}(X), MC_1)$ with the max available bandwidth.
9: let $Bandwidth(NeighborContainerPath_{server}) = max(Bandwidth(Path_{server}(Peer_{left}(X), MC_1)))$;
10: if $(Bandwidth(Path_{server}(X, MC_1)) \geq Bandwidth(NeighborContainerPath_{server}))$ then
11:     let $NeighborContainerPath_{server} = Path_{server}(x, MC_1)$;
12: end if
13: return $NeighborContainerPath_{server}$;
```

The procedure of meLBRouting is as follows: 1. Choose a neighboring container randomly as the next hop, and via it, compute a logical container path. In meGautz's $UK(2,m)$ Kautz topology, there are two node-disjoint paths by left- or right-shift operations, respectively. So meLBRouting can spread traffic along the different intermediate containers. 2. For building each physical inter-container sub-path, pick the servers and switches, among which the links have the most available bandwidth. Between the containers, there are plenty of L_{switch} and $L_{container}$ links, which can be utilized by meLBRouting to construct server- and switch-disjoint paths for balancing loads. This step is crucial and its pseudo-code is listed above. The path from the source to destination server in the next-hop containers comprises three sub-paths, and its bandwidth is determined by the link with the minimum bandwidth. To compute the bandwidth of each path, the server keeps probing the available bandwidth of links from it to its peer-switch servers periodically. This procedure is analogous to the fault-routing routing algorithm; the difference is that meLBRouting needs to find a path with the most available bandwidth, but that is a survival one. meGautz detects failures of links, servers and switches by means of period probing. It marks the bandwidth of failed paths as 0, and computes another available one to handle failures. More complex scenarios, like robust meRouting and meLBRouting under *churn* [22], will be studied in our future work.

We have the following Theorem for the meLBRouting algorithm.

Theorem 3. Consider a scenario that the servers in any two containers are under an all-to-all traffic pattern. With meLBRouting, the number of flows carried on a normal links L_{switch} is $\frac{N(4+A_s)}{4}$, and the number of flows carried on a virtual link VL is $\frac{N^2}{4}$.

From Theorem 3, we get the ratio of capacity of virtual links over that of normal links, $r = \frac{N}{4+A_s}$. We also take the example in last section, $N = 1,536$ and $A_s = 4.2$, and $r = 187.3$. Thus, for the case of two containers, the ABT is also constrained by normal links. Even if some high-speed links break down, it will not hurt the network throughput. Since meLBRouting also realizes the traffic isolation, $ABT = \frac{8N}{2+A_s} + \frac{8N}{A_{SCautz}}$.

5 Experiments and Analysis

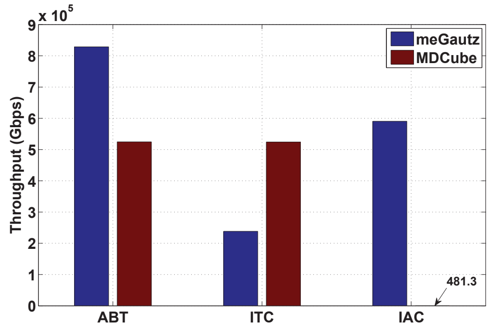
In this section, we evaluate meGautz from the following aspects: 1) the overall capacity of meGautz to process all-to-all data shuffle with inter- and intra-container traffic isolated. 2) throughput of multiple containers for all-to-all traffic, which reflects its capacity to support OLDI applications in the real world. 3) the performance fluctuation when failure occurs and increases. Through these experiments, we analyze and show the effect of traffic isolation on performance and reliability improvements for MDCN.

The most closely related solution to meGautz is MDCube [9], the state-of-the-art inter-container, server-centric datacenter network design that performs best in network capacity and fault-tolerance among existing designs. We take MDCube as a primary object to be compared with meGautz. Compared with meGautz, the topology of MDCube does not support strict traffic isolation of inter- and intra-container routing, therefore meGautz outperforms MDCube in all experiments where the traffic traverses both inter- and intra-container links. We assume that the intermediate servers relay traffic without delay, and do not consider the impacts of CPU on packet forwarding [23]. We simulate meGautz and MDCube by modifying the DLG simulator [24], and each experiment is conducted 20 times to calculate the average results.

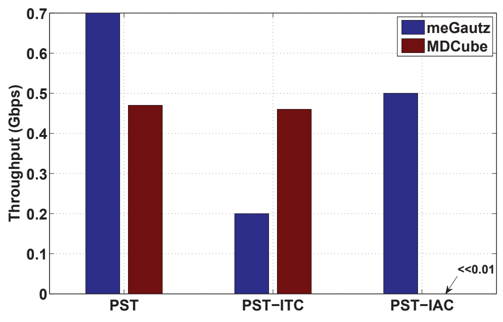
We use a meGautz built from 768 $SCautz(2,10,5)$ containers for mega-datacenters, and all the containers form a $UK(2,9)$ Kautz topology. In each container, 1,536 servers are directly interconnected into a $UK(2,10)$ Kautz topology, and 96 switches of two types form 96 "clusters" with 48 servers in each one. So there are 1,179,648 servers and 73,728 switches, in total. In meGautz, the servers are equipped with three dual-port Gigabit-NICs, and the switches have 48 1Gbps ports and four 10Gbps ports. For comparisons, we choose a typical 2D MDCube, recommended by [9]. It is constructed by 1,089 $BCube(32,1)$ containers [9]. Each container has 1,024 servers and 64 switches. So there are 1,115,136 servers and 69,696 switches in it.

TABLE 1: Key simulation results of meGautz and MDCube (Gbps).

	ABT	*ITC*	*IAC*	*PST*	$PST - ITC$	$PST - IAC$
meGautz	828,874.9	238,312.7	590,562.2	0.70	0.20	0.50
MDCube	524,656.5	524,175.2	481.3	0.47	0.46	$\ll 0.01$

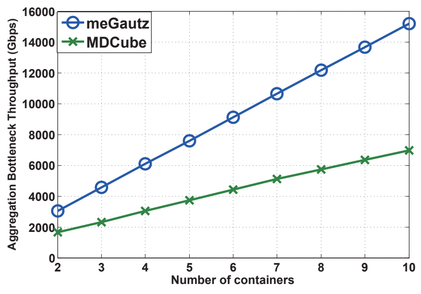


(a) Overall throughput

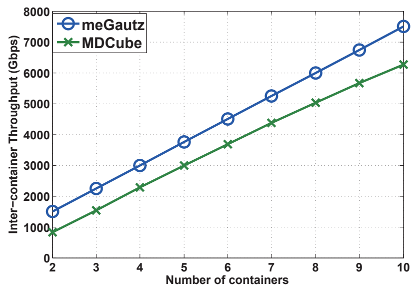


(b) Per-server throughput

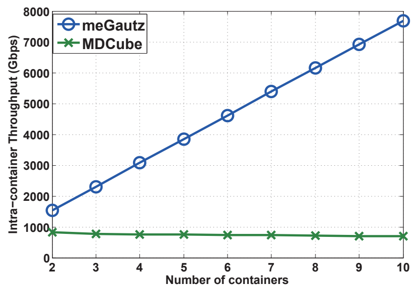
Fig. 4: The key metric of meGautz (with the basic meRouting algorithm) and MDCube under the all-to-all traffic pattern.



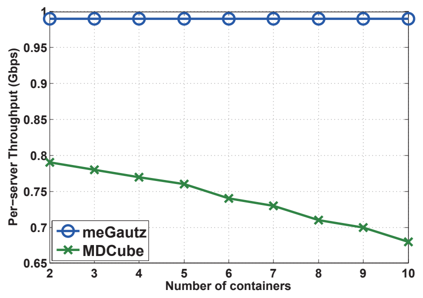
(a) ABT



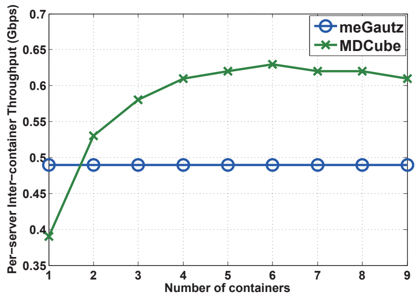
(b) ITC



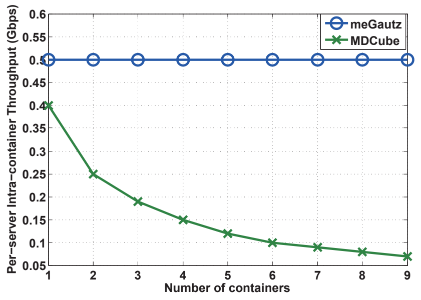
(c) IAC



(d) PST



(e) PST-ITC



(f) PST-IAC

Fig. 5: The throughput comparison between meGautz (with the load-balanced meLBRouting algorithm) and MDCube.

The quantities of containers and servers in the candidate meGautz and MDCube structures are both different. So, besides the well-known metrics ABT (Aggregation Bottleneck Throughput) [8], we also use PST (Per-server Throughput) [8], [16] to represent the capacity of each server for processing traffic bursts on average. Moreover, the inter- and intra-container throughput (ITC and IAC), and the per-server throughputs for inter- and intra-container traffic (PST-ITC, and PST-IAC) are introduced.

At last, we also analyze the cost of meGautz to build MDC at different scales, and compare with MDCube. The comparison shows that meGautz's cost is relatively low and suitable to deploy in practice.

5.1 Overall Performance with Traffic Isolated

Assuming that all the servers are under an all-to-all traffic pattern, we measure the overall performance of meGautz using the basic traffic-isolated *meRouting* algorithm. Compared to MDCube, we make some important observations, as follows.

First, as shown in Table 1 and Figure 4, the whole meGautz ABT is 828,874.9 Gbps and its PST is 0.70Gbps, which are improved by 58.0% and 48.9% over MDCube, respectively. Second, in meGautz, the ITC and IAC are 238,312.7 Gbps and 590,562.2 Gbps, which account for 28.8% and 71.2% of the total ABT. Its IAC is much higher than MDCube's IAC. More importantly, the average IAC of each container in meGautz is 768.9Gbps, which equals to the ABT of a separate $SCautz(2, 10, 5)$ container for processing its own all-to-all traffic, independently. While for

MDCube, its total IAC is just 481.3 Gbps, which accounts for far below 0.01% of its ABT. Even worse, its average IAC of each container is only 0.5 Gbps, while the ABT of a separate *BCube*$(32, 1)$ container is 1,057.1 Gbps [8]. Obviously, MDCube's inter-container traffic exhausts the most available bandwidth, and almost starves its intra-container traffic.

Thus, meGautz is able to provide much higher capacity than MDCube. In the meantime, it prevents the inter- and intra-container traffics from interrupting each other. In particular, the IAC of each container in meGautz can be achieved as high as ABT in a separate SCautz container. meGautz's traffic isolation effectively eliminates the overhead caused by resource competition, and so it achieves high performance of both of inter- and intra-container networks, and further that of the entire MDCN and individual servers.

5.2 Performance on OLDI support

OLDI applications generate massive all-to-all flows over multiple containers in MDCN. We designed a load-balanced *meLBRouting* algorithm to deal with these bursts efficiently. In this subsection, we evaluate its performance on shuffling the all-to-all traffic in two to ten containers.

From Figure 5(a) and 5(d), we know that meGautz's ABT grows linearly from 3,036.4 Gbps to 15,182.3 Gbps as the containers increase from two to ten. In the meantime, its PST is achieved and retained up to 0.99 Gbps. Clearly, the meLBRouting algorithm optimizes the resource utilization of meGautz and improves the performance. By comparison, the MDCube's ABTs are just 1,637.6 to 6,963.2 Gbps, when it uses its load-balancing routing algorithm to deal with the traffic bursts. Moreover, its PST drops from 0.79 to 0.68 Gbps, which are less than meGautz's by 25.3% and 45.6%, respectively. The reasons why MDCube suffers performance loss, but meGautz does not, are as follows.

On one hand, as implied by Figure 5(b) and 5(e), the ITCs of meGautz and MDCube both grow linearly, in which the ITC of meGautz (using the improved *meLBRouting* algorithm) is higher than that of MDCube. But meGautz's PST-ITC is kept at 0.49 Gbps without variations, while MDCube's increases sharply from 0.39 to 0.61 Gbps by 56.4%. In the simulations, we choose a typical 2D MDCube, which is constructed by *BCube*$(32, 1)$-containers, to compare to meGautz. Each BCube-container has four neighbors with the high-speed ports of switches linked directly. By using its detour routing algorithm [9] to balance load, the most servers can deal with the inter-container traffic via the neighboring containers. Therefore, MDCube's ITC is able keep increasing when the number of containers is not more than four, for there is little resource competition between the inter- and intra-container traffic. Once the number of containers is beyond four, the resource competition in MDCube becomes more frequent. The resulting performance loss will make the PST-ITC of MDCube flatten out.

On the other hand, as the number of containers rises to ten, Figure 5(c) and 5(f) shows that meGautz has its IAC to grow linearly, with the average IAC of each container (768.9 Gbps) and PST-IAC (0.50 Gbps) unchanged. However, although the number of servers rises from 2,048 to 10,240, MDCube's total IAC decreases, instead. It is because that the PST-IAC drops notably from 0.40 to 0.07 Gbps by 82.5%.

In summary, MDCube has the majority of resources overwhelmed by its inter-container flows, resulting in a significant overhead by resource competition. While in meGautz, meLBRouting not only makes flows utilize all the resources efficiently, but also achieves traffic isolation between the inter- and intra-container flows, and allocates bandwidth to them evenly. Thus, meGautz is able to provide a high and sustaining capacity to distribute traffic bursts quickly, and support OLDI services better.

5.3 Robustness

In this subsection we evaluate the robustness of meGautz under server/switch failures.

For the server-centric DCNs, server failures not only make computation and storage capacities decrease, but also hurt network performance. So, network capacity should not degrade faster than the computation and storage. In these simulations, we assume that all the faults are caused by servers and switches, and their links become unavailable. We study the throughput changes of two containers in meGautz and MDCube, as faults occur and increase. This scenario is ordinary and representative in practice, because an OLDI application always needs more than one container.

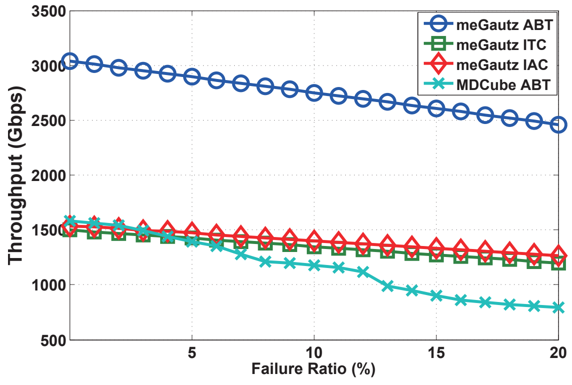
We first evaluate the impact of server failures on the throughput of meGautz, and compare it with MDCube under the same configuration. We emulate the failures by choosing failed servers randomly and uniformly from the network and making them not participate in the routing. The percentage of failed servers ranges from 0 to 20 percent. Typically, when 10% and 20% of servers fail, the computation capacity of datacenters drops by 10% and 20%. While for MDCN, the meGautz's ABT drops by 10.4% and 23.4%, in which the ITC decreases by 11.4% and 24.9%, and the IAC decreases by 9.7% and 21.9%, shown in Figure 6 (a). However, MDCube's ABT drops by 25.5% and 50.1%. Since our simplified simulator cannot distinguish the inter- and intra-container traffic of MDCube, Figure 6 depicts ITC and IAC only for meGautz.

We then evaluate the impact of switch failures on the throughput of meGautz, and compare it with MDCube under the same configuration. We emulate the failures by choosing failed switches randomly and uniformly from the network and making them not participate in the routing. The percentage of failed switches ranges from 0 to 20 percent. From Figure 6 (b), we see that 10% or 20% of failed switches result in only 4.78% and 9.73% ABT degradation of meGautz. It is because that the switch faults have little impact on its IAC, and consequently, much less impact on the network performance than servers. By comparison, MDCube's ABT shrinks over 50% when just 10% of the switches fail.

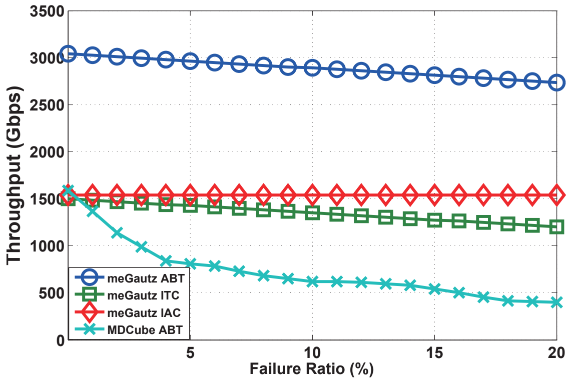
Thus, meGautz not only behaves much better than MDCube on tolerating the faults of servers and switches, but also makes the network performance degrade as slowly as the computation and storage do.

5.4 Cost comparison

Construction cost is an important issue, which needs to be considered for interconnecting containers to build MDC. meGautz and MDCube adopt the containers with different intra-container networks, and organize them into the different topologies, respectively. Consequently, the number of servers in SCautz- and BCube-containers are different, and the number of containers in a complete meGautz and MDCube are also different. To compare fairly, we estimate their cost to construct MDC with the same number of servers in a incomplete structure, including the number



(a) Server failures



(b) Switch failures

Fig. 6: The throughput degradation of two containers in meGautz and MDCube as failures occur and increase.

TABLE 2: Costs of meGautz and MDCube at various scale.

	Container NO.	*Swtich NO.*	*Links NO.*
small meGautz	7	672	2,688
small MDCube	10	640	2,560
medium meGautz	66	6,336	25,344
medium MDCube	98	6,272	25,088
large meGautz	652	62,592	250,368
large MDCube	977	62,528	250,112

of containers, switches, and links between containers they need at least.

Assuming the small, medium, and large scale MDC have 10,000, 100,000, and 1,000,000 servers, respectively, we compute and compare the construction costs of meGautz and MDCube. Typically, meGautz and MDCube still choose $SCautz(2,10,5)$- and $BCube(32,1)$-containers with same switches as in above three evaluations. As shown in Table 2, both meGautz and MDCube need a large number of switches and links to support efficient all-to-all traffic. On the other hand, meGautz and MDCube have almost the same costs. Although the number of switches and links in meGautz is slightly more than that in MDCube, the needed number of containers is less by about 30%. It helps lowering wiring and management complexity. The spare switches we need have already been prefabricated in each SCautz-container, and the number is fixed. In SCautz, they are reserved to deal with traffic bursts and frequent failures. Because meGautz does not introduce new switches, and the required optical fibers for container interconnections are slightly more than MDCube, so the cost for meGautz to construct massive datacenters is relatively low. Besides, meGautz achieves higher performance and better fault-tolerance than MDCube, so it is more cost-efficient.

6 Related works

The modular datacenter network (MDCN) allows the intra- and inter-container networks to be designed separately, which simplifies the construction and maintenance of huge MDC. It is considered as the ideal next-generated DCN, and lots of novel solutions [8], [9], [14], [16], [25], [26] have been proposed. The classic intra-container networks include CamCube [27], BCube [8], and SCautz [16]. CamCube adopts a direct-connect 3D torus topology, and it focuses mainly on providing flexible routing APIs for cloud applications. BCube interconnects servers into a hypercube topology, by taking the COTS switches as dumb crossbars; meanwhile, SCautz first connects the NIC ports of servers, forming a base Kautz topology, and then over-provides tens of COT switches, further building two logical Kautz structures. Both of them achieve an uniformly high network capacity and graceful performance degradation, in which SCautz behaves much better with lower cost, particularly, in fault-tolerance. SCautz and meGautz both successfully achieve high performance and fault tolerance. But SCautz only focuses on the flows in one container, while meGautz simultaneously deals with the inter- and intra-container flows effectively. meGautz is built based on SCautz for its special hierarchical structure, other than BCube, DCell, and Fat-tree. SCautz has a physical and two logical Kautz topologies. So it can accommodate a specific number of spare switches and links, which are used by meGautz to isolate traffic and tolerate faults. Therefore, meGautz is able to achieve higher throughput without resource competition, and fewer performance degradation with continuous failures.

In this paper, we focus on the inter-container networks. Helios [14] deploys both commercial 10Gbps packet switches and MEMS-based optical circuit switches between containers, and uses them to deal with the different communication patterns. To lower the cost, it tries to achieve full bisection bandwidth by allocating bandwidth dynamically. Facing the all-to-all traffic, the throughput of Helios is limited by its performance bottlenecks and worse than fat-tree. Jellyfish [28] adopts a degree-bounded random graph topology to interconnect the top-of-rack (TOR) switches. It realizes incremental expansion of MDCN to build massive MDC with different degrees of oversubscription. For one-to-one traffic pattern, Jellyfish is more cost-efficient than a fat-tree, while for all-to-all, its throughput will be limited by the bottlenecks. uFix [15] builds mega-datacenters based on the heterogeneous containers, by interconnecting just one Gigabit-NIC port of servers. Therefore, its aggregation bandwidth between containers is small, resulting in a poor per-server throughput of 0.21 Gbps at most. By contrast, MDCube [9] connects the neighboring containers via two switches, and arranges containers into a 1D or 2D hypercube structure. Therefore, its capacity and reliability are both better than Helios, Jellyfish, and uFix. However, as in uFix, all its links are used by both of BCube and MDCube simultaneously. So, no matter if the intra- or inter-container traffic increases, it will preempt the available bandwidth, resulting in a significant throughput decline of the other one. Moreover, switches between containers are the single point of failure in MDCube, i.e., the failure of only one switch will disconnect the neighboring containers. Despite it still being reach-

TABLE 3: Comparison of different network structures.

	Degree	Diameter	Bisection Bandwidth
hypercube	$\log_2 N$	$\log_2 N$	$\frac{N^2}{4}$
d-torus	$2d$	$\frac{1}{2dN^{\frac{1}{d}}}$	$\frac{1}{4dN^{\frac{1}{d}}}$
fat tree	/	$2log_2N$	$N/2$
Kautz	d	$\log_d N - \log_d(1+1/d)$	$\frac{2dN}{\log_d N - \log_d(1+\frac{1}{d})}$

able via other intermediate containers, the fault-tolerant routing will take more hops and consume extra bandwidth, making the performance of MDCube decrease sharply. For meGautz, there are three advantages in gaining higher throughput over MDCube: topology design, construction method, and traffic-isolated routing algorithm. First, we design meGautz based on a constant-degree Kautz graph. The Kautz graph can obtain a smaller diameter than other topologies with the same degree and order, e.g., hypercube in MDCube, Butterfly, and d-dimensional torus. Second, we make a tradeoff between the degree and diameter of meGautz. The rich connectivity and high bandwidth between the containers support OLDI services well. Third, the inter- and intra-container routing makes full use of the different links and switches to forward their flows. The inter- and intra-container traffic are isolated to avoid throughput loss caused by resource competition.

Traffic isolation is firstly considered as a design goal of MDCN, has been well achieved by meGautz to avoid performance loss caused by resource competition. As discussed in Section 2, there are two way for container interconnection: introducing new high-end switches [14] and wiring the existed components [9], [15], [28]. The former uses the new switches and external links transfer the inter-container flows, which do not have to rely on any links inner the containers. For example, MDCN is built by wiring containers with core switches into a fat-tree topology. To be non- oversubscribed, its number of servers in each pod or container, and links and switches in each layer are all accurately planed. Every inter-container (or inter-pod) flows must be evenly mapped to the different paths and be forwarded by one core switch, needless to get a third container (or pod) involved. So, the MDCN can achieve full bisection bandwidth for there is no resource competition. The latter wires the free components in the different containers, e.g., uFix wiring NICs of servers, MDCube and Jellyfish wiring high-speed ports of switches. Before reaching the destination server, an inter-container flow has to traverse multiple intermediate containers, entering from one port and leaving from another. If it has to multiplex the resources in containers, resource competition will occur, no matter that the MDCN is switch- or server-centric. Take MDCube as an example, it has high-speed ports of the different switches in each BCube-container linked to the neighboring containers. MDCube computes the sub-paths through the intermediate containers for inter-container flows, by leveraging the same routing algorithm and links in BCube. Therefore, when MDCube's servers have an all-to-all traffic from two to ten containers each with 1,024 servers, its per-server throughput for inter-container traffic (PST-ITC) increases from 0.39 to 0.61 Gbps by 56.4%; whereas its per-server throughput for intra-container traffic (PST-IAC) decreases from 0.40 to 0.07 Gbps by 82.5%. As a result, the total per-server throughput (PST) of MDCube drops by 16.2%, from 0.79 to 0.68 Gbps. Since MDCube is over-subscripted and the bottleneck is its intra-container links [9], their bandwidth is continuously preempted by inter-container flows and links get more congested, as the number of containers grows. So MDCube's PST-IAC is reduced obviously, and its PST decreases as well. In sum, all the existed inter-container networks dont consider the problem of resource contention. They leverage the intra-container links to realize inter-container routing, and allocate bandwidth to the intra- and inter-container flows proportional to their quantities. If the novel routing algorithms are proposed to limit the two kinds of flows at computed rates by switches and servers, the available network capacity can be utilized optimally. So either kind of flows cannot consume excessive resources, making the other one starve. But if the inter- and intra-container flows are still transferred via one or several same links in the containers, e.g., in Jellyfish [28], the resource competition can also not be eliminated. By comparison, meGautz is able to isolate traffic from SCautz, by using the spare switches and links in containers. So it eliminates the resource competition completely.

We design meGautz based on Kautz graphs [17], [29]. As Table 3 shows, among all the existing graphs (including hypercube, d-torus, fat tree, etc.), Kautz graphs have the smallest diameter $D = log_d N$ (given the number of nodes N and maximum degree d), which reaches the Moore bound [30]; Kautz graphs also accommodate the maximum number of nodes of $d^D + d^{D-1}$ (given the diameter D and maximum degree d). Furthermore, Kautz graphs have many other desirable properties such as maximum connectivity, constant congestion and robust routing.

meGautz realizes traffic isolation from the topology perspective, i.e., the inter- and intra-container routing is conducted using different links of the topology. Therefore, our design is orthogonal to packet-level isolation methods like MPLS (Multi-Protocol Label Switching) [31], traffic classification [32], and bandwidth provisioning [33]. For example, one can realize inter- and intra-container traffic isolation on a meGautz network, and further distinguish intra-container traffic based on the label of each packet [31].

Additionally, there are a lot of ongoing works about MDCN in other aspects, such as energy saving, TCP in datacenters [34], flow scheduling [35], and network virtualization [7]. Our work is orthogonal, and takes some inspiration from them.

7 Conclusion and Future Work

According to the distinctive prosperities of MDC, inter-container networks should not only be high capacity and fault-tolerant, but also should isolate traffic from the intra-container networks. In this paper, we have presented the design, construction, and evaluation of meGautz. By making full use of the spare resources in containers, it isolates traffic between the inter- and intra-container routing, so that its intra-container throughput of each container can be achieved as high as the throughput of an individual SCautz-container; its total throughput is 40% more than MDCube. As far as we know, in the server-centric inter-container networks, meGautz is the first to realize traffic isolation in both topology and routing algorithms. Due to rich connectivity between neighboring containers and multiple paths in meGautz, the reliability of the entire MDCN is improved greatly. Our theoretical analysis and experiments show that meGautz is an attractive and valid inter-container network structure for MDC. In the ongoing work, we are building a practical prototype system to further evaluate its characteristics.

In this paper we mainly focus on all-to-all traffic, which is the *worst-case* communication pattern in datacenter networks [36].

As discussed in [13], however, some OLDI applications' traffic is not always all-to-all. Therefore, it is possible to design simplified network topologies that could satisfy specific OLDI applications (but provide less ABT for all-to-all traffic than meGautz), which will be studied in our future work. The incremental deployment of meGautz is another problem we plan to solve in the future work. In [37], the author has presented a method for constructing incremental scalable partial line diagraphs [38] of Kautz graphs with best possible connectivity, close to, or exactly equal to, d. It makes the Kautz graphs unique in the sense that they are not only the densest digraphs, but also incremental scalable. Moreover, the topology expanding of Kautz graphs have been deeply studied in DHT of P2P [29], [30]. We will take some inspiration from them and realize incremental scalability of meGautz without rewiring links between containers.

Acknowledgments

This work is supported in part by the National Basic Research Program of China (973) under Grant No. 2014CB340303, the National Natural Science Foundation of China (NSFC) under Grant No. 61379055, 61379053, and 61222205, the Program for New Century Excellent Talents in University, and the Fok Ying-Tong Education Foundation under Grant No. 141066.

References

[1] K. V. Vishwanath, A. Greenberg, and D. A. Reed, "Modular data centers: how to design them?" in *Proc. of LSAP*, New York, USA, 2009.

[2] J. R. Hamilton, "Architecture for modular data centers," in *3rd CIDR*, 2007.

[3] Rackable systems. ice cube modular data center. [Online]. Available: http://www.rackable.com/products/icecube.aspx

[4] Cisco modular datacenter. [Online]. Available: http://www.cisco.com/en/US/netsol/ns1132/index.html

[5] Google modular datacenter. [Online]. Available: http://www.google.com/about/datacenters/index.html

[6] A. Hammadi and L. Mhamdi, "A survey on architectures and energy efficiency in data center networks," *Computer Communications*, vol. 40, pp. 1–21, 2014.

[7] M. F. Bari, R. Boutaba, R. Esteves, L. Z. Granville, M. Podlesny, M. G. Rabbani, Q. Zhang, and M. F. Zhani, "Data center network virtualization: A survey," *Communications Surveys & Tutorials, IEEE*, vol. 15, no. 2, pp. 909–928, 2013.

[8] C. Guo, G. Lu, D. Li, H. Wu, X. Zhang, Y. Shi, C. Tian, Y. Zhang, and S. Lu, "BCube: a high performance, server-centric network architecture for modular data centers," in *Proc. of SIGCOMM*, New York, USA, 2009, pp. 63–74.

[9] H. Wu, G. Lu, D. Li, C. Guo, and Y. Zhang, "MDCube: a high performance network structure for modular data center interconnection," in *Proc. of CoNEXT*, New York, USA, 2009, pp. 25–36.

[10] K. Chen, X. Wen, X. Ma, Y. Chen, Y. Xia, C. Hu, and Q. Dong, "WaveCube: A scalable, fault-tolerant, high-performance optical data center architecture," 2015.

[11] Y. Zhao, J. Wu, and C. Liu, "On peer-assisted data dissemination in data center networks: Analysis and implementation," *Tsinghua Science and Technology*, vol. 19, no. 1, pp. 51–64, 2014.

[12] D. Meisner, C. M. Sadler, L. A. Barroso, W. Weber, and T. F. Wenisch., "Power management of online data-intensive services," in *Proc. of ISCA*, 2011.

[13] B. Vamanan, J. Hasan, and T. N. Vijaykumar., "Deadline-aware datacenter tcp (d2tcp)," in *Proc. of SIGCOMM*, 2013.

[14] N. Farrington, G. Porter, S. Radhakrishnan, H. H. Bazzaz, V. Subramanya, Y. Fainman, G. Papen, and A. Vahdat, "Helios: a hybrid electrical/optical switch architecture for modular data centers," in *Proc. of SIGCOMM*, New York, USA, 2010, pp. 339–350.

[15] D. Li, M. Xu, H. Zhao, and X. Fu, "Building mega data center from heterogeneous containersserverswitch: a programmable and high performance platform for data center networks," in *Proc. of ICNP*, 2011, pp. 256–265.

[16] F. Huang, X. Lu, D. Li, and Y. Zhang, "SCautz: a high performance and fault-tolerant datacenter network for modular datacenters," *Science China Information sciences*, vol. 55, no. 7, pp. 1493–1508, 2012.

[17] L. Bhuyan and D. Agrawal, "Design and performance of generalized interconnection networks," *IEEE Transactions on Computers*, pp. 1081–1090, 1983.

[18] G. Wang, D. G. Andersen, M. Kaminsky, K. Papagiannaki, T. E. Ng, M. Kozuch, and M. Ryan, "c-Through: part-time optics in data centers," in *Proc. of SIGCOMM*, New York, USA, 2010, pp. 327–338.

[19] J. Li, D. Li, Y. Ye, and X. Lu, "Efficient multi-tenant virtual machine allocation in cloud data centers," *Tsinghua Science and Technology*, vol. 20, no. 1, pp. 81–89, 2015.

[20] F. Huang, D. Li, K.-L. Tan, J. Wu, and X. Lu, "meGautz: a high capacity, fault-tolerant and traffic isolated inter-container datacenter network," available at http://www.973ivce.org/hf/megautz.pdf, Tech. Rep., 2012.

[21] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta, "VL2: a scalable and flexible data center network," in *Proc. of SIGCOMM*, New York, USA, 2009.

[22] P. Godfrey, S. Shenker, and I. Stoica, "Minimizing churn in distributed systems," in *ACM SIGCOMM*. ACM, 2006.

[23] G. Lu, C. Guo, Y. Li, Z. Zhou, T. Yuan, H. Wu, Y. Xiong, R. Gao, and Y. Zhang, "Serverswitch: a programmable and high performance platform for data center networks," in *Proc. of NSDI*, 2011.

[24] Y. Zhang and L. Liu, "Distributed line graphs: A universal technique for designing dhts based on arbitrary regular graphs," *Knowledge and Data Engineering, IEEE Transactions on*, vol. 24, no. 9, pp. 1556–1569, 2012.

[25] D. Guo, T. Chen, D. Li, Y. Liu, X. Liu, and G. Chen, "BCN: expansible network structures for data centers using hierarchical compound graphs," in *Proc.of INFOCOM*. IEEE, 2011, pp. 61–65.

[26] D. Guo and T. Chen, "Expansible and cost-effective network structures for data centers using dual-port servers," *IEEE Transactions on Computers*, vol. 62, no. 7, pp. 1303–1317, 2013.

[27] H. Abu-Libdeh, P. Costa, A. Rowstron, G. O'Shea, and A. Donnelly, "Symbiotic routing in future data centers," in *ACM SIGCOMM Computer Communication Review*, vol. 40, no. 4. ACM, 2010, pp. 51–62.

[28] A. Singla, C.-Y. Hong, L. Popa, and P. B. Godfrey, "Jellyfish: Networking data centers randomly." in *NSDI*, vol. 12, 2012, pp. 17–17.

[29] D. Li, X. Lu, and J. Wu, "Fissione: A scalable constant degree and low congestion dht scheme based on kautz graphs," in *Proc. of INFOCOM*, vol. 3. IEEE, 2005, pp. 1677–1688.

[30] Y. Zhang and L. Liu, "Distributed line graphs: A universal technique for designing dhts based on arbitrary regular graphs," *Knowledge and Data Engineering, IEEE Transactions on*, vol. 24, no. 9, pp. 1556–1569, 2012.

[31] C. Guo, G. Lu, H. J. Wang, S. Yang, C. Kong, P. Sun, W. Wu, and Y. Zhang, "Secondnet: a data center network virtualization architecture with bandwidth guarantees," in *Proceedings of the 6th International COnference*. ACM, 2010, p. 15.

[32] J. Erman, M. Arlitt, and A. Mahanti, "Traffic classification using clustering algorithms," in *Proceedings of the 2006 SIGCOMM workshop on Mining network data*. ACM, 2006, pp. 281–286.

[33] L. Popa, P. Yalagandula, S. Banerjee, J. C. Mogul, Y. Turner, and J. R. Santos, "Elasticswitch: practical work-conserving bandwidth guarantees for cloud computing," in *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 4. ACM, 2013, pp. 351–362.

[34] C. Wilson, H. Ballani, T. Karagiannis, and A. Rowtron, "Better never than late: Meeting deadlines in datacenter networks," *SIGCOMM-Computer Communication Review*, vol. 41, no. 4, p. 50, 2011.

[35] A. Curtis, J. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee, "Devoflow: Scaling flow management for high-performance networks," *SIGCOMM-Computer Communication Review*, vol. 41, no. 4, p. 254, 2011.

[36] H. Ballani, P. Costa, T. Karagiannis, and A. Rowstron, "Towards predictable datacenter networks," in *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 4. ACM, 2011, pp. 242–253.

[37] P. Tvrdik, "Factoring and scaling kautz digraphs," *Citeseer*, 1994.

[38] M. Fiol, A. S. Lladó *et al.*, "The partial line digraph technique in the design of large interconnection networks," *Computers, IEEE Transactions on*, vol. 41, no. 7, pp. 848–857, 1992.

Feng Huang received his PhD degree in computer science and technology from the College of Computer, National University of Defense Technology in 2013. His research interests include cloud computing, data center networking, virtualization technology, and high-performance router.

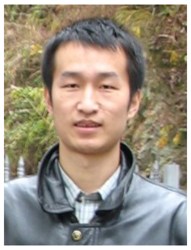
Yiming Zhang is currently an associate professor in the National Laboratory for Parallel and Distributed Processing, College of Computer, NUDT. His current research interests include cloud computing and operating systems. He received the China Computer Federation (CCF) Distinguished PhD Dissertation Award in 2011 and the HP Distinguished Chinese Student Scholarship in 2008. His current research is primarily sponsored by NSFC.

Dongsheng Li is a professor and doctoral supervisor in the College of Computer at National University of Defense Technology (NUDT). He received his PhD degree in computer science and technology from NUDT in 2005. He was awarded the Chinese National Excellent Doctoral Dissertation in 2008. His research interests include distributed systems, cloud computing and big data processing.

Jiaxin Li is a PhD candidate in the College of Computer at National University of Defense Technology (NUDT). He received his Master's Degree in computer science and technology from NUDT in 2013. His research interests include distributed systems, and computer networks & communications.

Jie Wu is a professor in the College of Science and Technology at Temple University. He received his PhD degree in computer engineering from Florida Atlantic University in 1989. His research interests include mobile computing and wireless networks, routing protocols, computer and network security, distributed computing, and fault-tolerant Systems.

Kaijun Ren is a professor at National University of Defense Technology (NUDT). He received his PhD degree in computer science and technology from NUDT in 2008. His research interests include Web-based systems, cloud computing and big data processing.



Deke Guo received his B.E. degree in Department of Industry Engineering from Beijing University of Aeronautic and Astronautic, and his Ph.D. degree in School of Information System and Management, National University of Defense Technology. His research interests include multi-hop wireless networks and data center networking.

Xicheng Lu is a professor and doctoral supervisor in the College of Computer at National University of Defense Technology (NUDT). He is a member of the Chinese Academy of Engineering since 1999. His research interests include parallel and distributed processing, and computer networks.