Received November 15, 2014, accepted December 6, 2014, date of publication December 18, 2014, date of current version December 30, 2014.

Digital Object Identifier 10.1109/ACCESS.2014.2383439

Rethinking the Data Center Networking: Architecture, Network Protocols, and Resource Sharing

TING WANG[1], (Student Member, IEEE), ZHIYANG SU[1], (Student Member, IEEE), YU XIA[1], (Member, IEEE), AND MOUNIR HAMDI[1,2], (Fellow, IEEE)

[1]Department of Computer Science and Engineering, The Hong Kong University of Science and Technology, Hong Kong
[2]College of Science, Engineering and Technology, Hamad Bin Khalifa University, Doha, Qatar

Corresponding author: T. Wang (twangah@cse.ust.hk)

This work was supported by the National Priorities Research Programme under Grant NPRP 6718-2-298 through the Qatar National Research Fund.

ABSTRACT Large-scale data centers enable the new era of cloud computing and provide the core infrastructure to meet the computing and storage requirements for both enterprise information technology needs and cloud-based services. To support the ever-growing cloud computing needs, the number of servers in today's data centers are increasing exponentially, which in turn leads to enormous challenges in designing an efficient and cost-effective data center network. With data availability and security at stake, the issues with data center networks are more critical than ever. Motivated by these challenges and critical issues, many novel and creative research works have been proposed in recent years. In this paper, we investigate in data center networks and provide a general overview and analysis of the literature covering various research areas, including data center network interconnection architectures, network protocols for data center networks, and network resource sharing in multitenant cloud data centers. We start with an overview on data center networks and together with its requirements navigate the data center network designs. We then present the research literature related to the aforementioned research topics in the subsequent sections. Finally, we draw the conclusions.

INDEX TERMS Data center networks, architecture, network protocols, resource sharing.

I. INTRODUCTION

As computation progressively moves into the cloud, the data center has been playing a more and more important role in supporting the enterprise networks and cloud computing services, such as large-scale computations, web searching, email, online gaming, social networking, and so on. Typically, these services are provided by a shared data center because the physical equipment (like servers, network devices, cooling and humidification system, power supply system, lighting system, and so on.) and their maintenance are too expensive for tenants to build their own data centers. The data center network (DCN) connecting the servers plays a crucial role in orchestrating the data center to deliver peak performance services with strong reliability to users. In order to achieve this goal, service providers should make sure all the resources in data center must work in concert, which can be achieved by a holistic approach to their design and deployment. Particularly, cloud computing is characterized as the culmination of the integration of computing and data infrastructures to provide a scalable, elastic, agile and cost-effective approach to support the ever-growing critical IT needs of both enterprises and the general public.

As noted, large-scale data centers serve as the core infrastructure for the Cloud [21]. To support the growing cloud computing needs, the number of servers in today's data centers is increasing exponentially, thus resulting in enormous challenges to efficient network design for interconnecting these servers so that the deployment and maintenance of the infrastructure is cost-effective. The network is to a server cluster what the central nervous system is to the human body. With data availability and security at stake, the role of data center network is more critical than ever.

Generally, recent research in data center networks has addressed many of existing issues from various aspect. The researchers have proposed lots of creative ideas, such as creating large flat Layer 2 networks, designing near optimal

transport protocols, virtualizing network resources, and so on. Based on their previous contributions and current findings, the main contribution of this paper is that we conducted a comprehensive investigation in the data center networks, and studied the related works that have been proposed in recent years. Besides, based on careful observations we also advance some forward-looking thoughts and proposals on the existing issues of data center networking. The primary goal of this paper is to address the existing network challenges and feasible solutions that are associated with data center networking.

The rest of this paper is organized as follows. We start with an overview on the data center networking in Section 2. Next we turn to examine some typical representatives of DCN architecture that have been proposed in the research literature, and compare them from various aspects in Section 3. Thereafter, we focus on some novel proposals on the transport protocols in data center networks, which aim to provide efficient and reliable data transmission in data center network in Section 4. Then we discuss how to share the network resources and achieve bandwidth guarantee and performance isolations in multi-tenancy cloud data centers in Section 5. Finally, we draw a conclusion in Section 6, which concludes the paper.

II. OVERVIEW ON DATA CENTER NETWORKING

A data center (sometimes spelled *datacenter*) is regarded as a physical or virtual centralized repository for the computation, storage, management, and dissemination of information and data. As a centralized repository, a typical data center usually consists of computers, switches/routers, racks of servers (like web servers, application servers, database servers), load balancers, wire cages or closets, power distribution equipment, cooling and humidification system, lighting system, and other related equipments. The number of servers and other necessary devices in data centers are becoming increasingly large due to an increase in the uptake of cloud computing and cloud-based services. All these infrastructures within a data center are orchestrated by the data center network to work as an organic cohesive whole. In order to make the data center flexible and adapt to dynamic varying demands, both server virtualization and network virtualization are being considered to be employed.

A. DATA CENTER NETWORK REQUIREMENTS

The data center network connecting the servers in the data center plays an important role in orchestrating the data center to deliver peak performance to users. It not only determines the reliability of data centers, but also largely impacts network capacity, fault tolerance, latency, and routing efficiency. Generally, the design goals of data center networks are high scalability, good fault tolerance, low latency, high network capacity and well virtualized if necessary [7].

1) SCALABILITY

In order to meet the increasing demands for services and better performance, the physical structure must have good scalability enabling incremental expansion without affecting the existing servers. Correspondingly, the routing algorithm should also be scalable and easily adapt to the new expanded interconnection.

2) FAULT TOLERANCE

A fault-tolerant architecture allows the system to continue with its current task even in the presence of failures. From the perspective of the network design, good fault tolerance can be achieved through redundant physical connections and fault-tolerant routing algorithms. Besides, the detection of failures should be rapid and efficient.

3) LATENCY

Primarily the network latency consists of the queuing delay at each hop, transmission delay and propagation delay, of which the buffer queuing at each hop is the major contributor to latency. Therefore, a smaller network diameter, which leads to lowering the latency should be offered as a basic feature of the network design enabling the data center to provide faster services.

4) NETWORK CAPACITY

Large-scale data centers providing cloud services are usually bandwidth hungry. Hence, the data center should provide high network capacity to support the high volumes of traffic generated by many online infrastructure services, such as GFS [42] and MapReduce [28].

5) VIRTULIZATION

Any Virtual Machine (VM) may migrate to any physical machine with no need to change its IP address.

Additionally, performance isolation should also be well supported, which implies that traffic from one application should not be affected by others. To meet these challenging design goals, the researchers have proposed many creative approaches. These proposals will be discussed in the subsequent sections, which provide a detailed coverage of various research interests in data centers.

III. DATA CENTER NETWORK TOPOLOGIES

Data center network is constructed to better accommodate dynamic virtualized servers and storage environments, while the DCN architecture is regarded as one of the most important determinants of network performance, and it plays a significant role in meeting the requirements of could services as well as the agility and dynamic reconfigurability of the infrastructure for changing application demands. As a result, many novel proposals, such as Fat-Tree [1], OSA [30], DCell [7], BCube [22], VL2 [5], c-Through [20], Helios [37], FiConn [13], MDCube [24], HyperBcube [15], Portland [40], FlatNet [14], SprintNet [46], CamCube [2], Small-World [29], NovaCube [45] have been proposed aiming to efficiently interconnect the servers inside a data center to deliver peak performance to users.

Generally, the DCN topologies can be roughly classified into four categories: tree-based topology (e.g. Fat-Tree,

VL2, Portland), recursive-defined topology (e.g. DCell, BCube, FiConn, FlatNet, SprintNet), hybrid network (e.g. c-Through, Helios) and direct network (e.g. CamCube, Small-World). If considering whether the network topology is changed from the time it is deployed, the DCN topologies can be divided into fixed topology (such as Fat-Tree, Portland, VL2, DCell, BCube, etc) and flexible topology (like c-Through, OSA, Helios). From another perspective, the DCN architectures can also be divided into two groups: *server-centric* and *switch-centric*. Unlike the *switch-centric* scheme which places the interconnection and routing intelligence on switches, the *server-centric* scheme expects the servers also to forward packets. DCell, BCube, FiConn, FlatNet, SprintNet and HyperBcube are examples of *server-centric* architectures. In the following subsections we will conduct a detailed investigation in some typical representative network topologies, and compare them from various aspects.

A. TREE-BASED TOPOLOGIES

1) BASIC TREE

The traditional basic tree network topology shown in Fig.1 is usually built with two or three tiers, which are edge tier, aggregation tier (if necessary), and core tier. The servers are tree leaves, which are connected to many top of rack (ToR) switches. These ToR switches are then interconnected through end of rack (EoR) switches, which are in turn connected through core switches. The aggregation tier provides the domain service and load balancing, while the core tier interconnects the aggregation switches together and manages the traffic into and out of the data center. This kind of hierarchy approach results in a serious bandwidth oversubscription towards the network core tier. In response to this issue, a more cost-effective topology Fat-Tree was proposed.

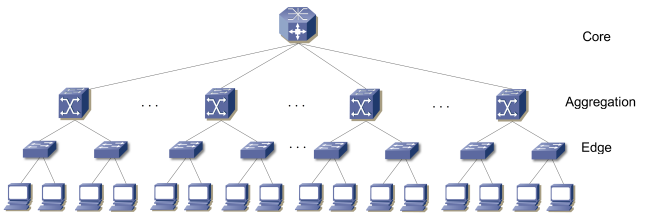


FIGURE 1. A traditional 3-level tree-based data center network topology.

2) FAT-TREE AND CLOS NETWORK

Fig.2 depicts a Fat-Tree topology, which is a folded-Clos network built in the form of multi-rooted tree. Fat-Tree topology was firstly introduced by Al-Fares et al. to construct the data center network [1]. Thereafter, Fat-Tree has been used as a typical topology for research on data center network by researchers, for example Portland [40], Hedra [32], Elastic Tree [26], and so on.

Fat Tree [1] is a rearrangeably non-blocking structure, which provides an oversubscription ratio of 1:1 to all servers. A Fat Tree built with n-port switches has n pods, each of which contain two layers of $n/2$ switches. It consists of $(n/2)^2$ core switches, $n^2/2$ aggregation and edge switches respectively, and supports $n^3/4$ servers in total. However, the wiring complexity is $O(n^3)$ which is a serious challenge.

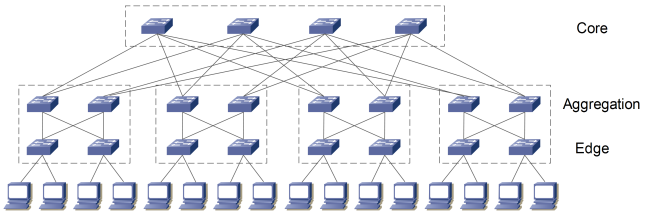


FIGURE 2. A simple 3-level Fat-Tree topology.

Portland [40] is a scalable, easily manageable, fault-tolerant, and efficient layer 2 Ethernet-compatible routing, forwarding and address resolution protocol for data center environments. It uses a pre-determined Fat-Tree topology, hierarchical pseudo-MAC (PMAC) address, and logically centralized directory (fabric manager or FM) to resolve PMAC to actual MAC (AMAC) addresses to enable Ethernet-compatible plug-n-play networking. Besides, based on multi-rooted Fat-Tree architecture, it exploits the knowledge of the underlying topology of a data center, and completely avoids broadcast-based mechanisms, and stresses on fast and easy virtual machine mobility within the data center. However, as a centralized system, the robustness and scalability of Fabric Manager is a critical issue since all functionalities rely on it. Besides, it also does not consider how to handle traffic congestion in case of multicast or high traffic volume.

3) VL2

VL2 [5] is an agile and cost effective network architecture, which is built from numerous switches arranged into a Clos topology. VL2 employs Valiant Load Balancing (VLB) to spread traffic across network paths, and uses address resolution to support large server pools. Besides, VL2 applies flat addressing to eliminate fragmentation of resources and allow any service to be assigned to any server anywhere in the data center. However, the directory system may become a bottleneck in the case of heavy network load.

B. RECURSIVE-DEFINED TOPOLOGIES

1) DCell

DCell [7], as shown in Fig.3(b), uses servers with multiple ports and low-end mini-switches to build its recursively defined architecture. In DCell, the most basic element named DCell_0 consists of n servers and one n-port switch. Each server in a DCell_0 is connected to the switch in the same DCell_0. Generally, DCell_k is created by $t_{k-1} + 1$ DCell_{k-1}s, where t_{k-1} is the number of servers in each DCell_{k-1}. The node degree of each server in a DCell_k is $k + 1$, and the level-i link connects to a different DCell_{i-1} within the same DCell_i. DCell scales out at a double exponential speed, and its fault-tolerant DFR routing protocol which introduces local-reroute achieves good results. However, the lower level links carry more traffic causing higher link utilization, thus may

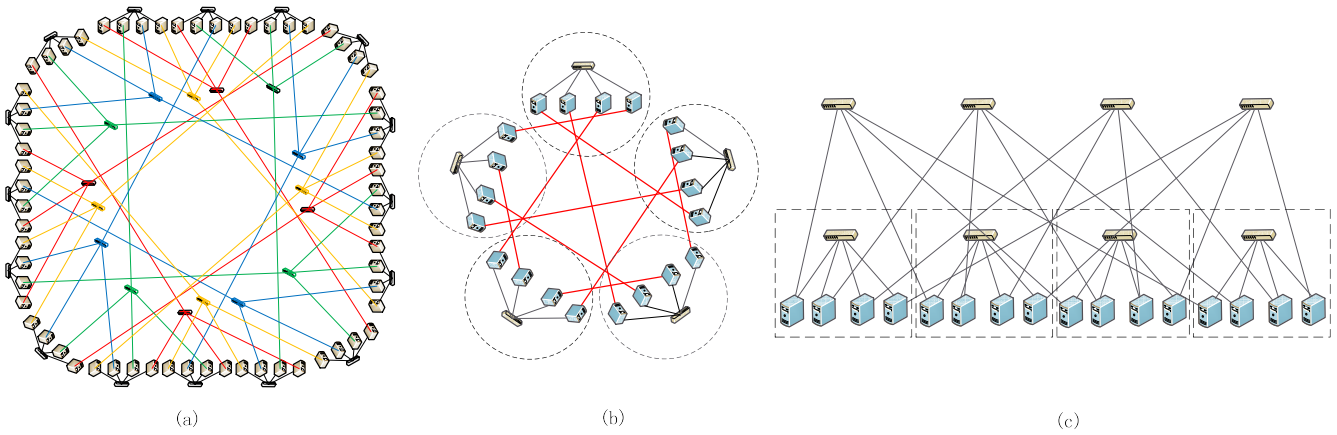


FIGURE 3. Examples of recursive topologies. (a) FlatNet. (b) DCell. (c) BCube.

become a bottleneck and result in low aggregate bottleneck throughput.

2) BCube

BCube [22], as shown in Fig.3(c), is also a recursively defined structure, which is specially designed for shipping container based modular data centers. Its most basic element BCube_0 is the same as DCell_0: n servers connect to one n-port switch. While constructing a BCube_1, n extra switches are used, connecting to exactly one server in each BCube_0. More generally, BCube_k is derived from n BCube_{k-1}s and n^k n-port COTS switches. Different from DCell, the servers in BCube only connect to switches without connecting other servers. BCube accelerates 1-to-n traffic patterns and also demonstrates a good network capacity for all-to-all network traffic. However, BCube is deficient in scalability with relatively high wiring complexity.

3) FiConn

FiConn [13] shares similar design principle as DCell. The main difference is that the degree of each server in FiConn is always two while in DCell it is $k+1$. Likewise, a high-level FiConn is derived from number of low-level FiConns. And the routing algorithm in FiConn is traffic-aware which can help it in utilizing the link capacities referring to traffic states, resulting in a good aggregate throughput. However, the fault tolerance and network capacity in FiConn are not so good. Besides, the average path length is relatively long.

4) FlatNet

FlatNet [14] is a 2-layer server-centric architecture, which scales at a speed of n^3. Briefly, the first-layer of FlatNet consists of n servers which connect to one n-port switch. And the second-layer of FlatNet is constructed by n^2 1-layer FlatNets. Different subsystems (1-layer Cells) are interconnected to each other using extra n^2 n-port switches. Fig.3(a) presents an example of a 64-server FlatNet architecture where $n = 4$.

5) SprintNet

Fig.4 shows an example of a 20-server SprintNet constructed by using 6-port switches when $c = 2$, $n = 6$ and $k = 1$. Each *Cell* is composed of 2 switches and 4 servers. The basic building unit of SprintNet is named *Cell* (or 0-layer), which is the building block to construct a larger SprintNet. Each *Cell* is constructed with c n-port switches, where $\frac{c}{c+1}n$ ports of each switch connect to $\frac{c}{c+1}n$ servers and $\frac{1}{c+1}n$ ports for inter-*Cell* connections. Accordingly, each *Cell* contains c n-port switches and $\frac{c}{c+1}n$ servers. All the switches and servers are fully-connected. The higher k-layer SprintNet is constructed by adding $\frac{c}{c+1}n$ *Cell*s each time and fully connected to each other in the same way. SprintNet is highlighted by its low diameter, good fault tolerance, and high aggregate bottleneck throughput. The specially designed traffic-aware adaptive and fault tolerant routing scheme helps SprintNet achieve its maximum theoretical performance. However, its scalability is relatively low.

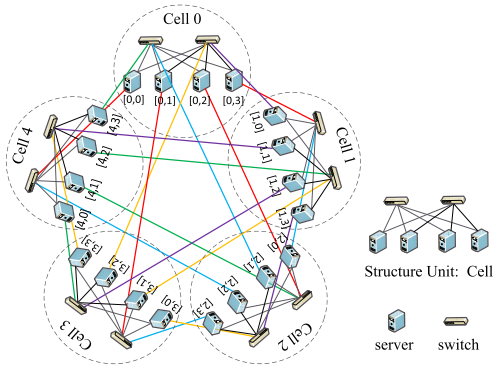


FIGURE 4. A 20-server *SprintNet* using two 6-port switches in each Cell.

6) ANALYSIS

The recursive-defined topologies are almost server-centric architectures, where the computational servers are not only running regular applications but also acting as routers to relay

the traffic in the system. This brings numerous advantages and convenience, but it is also accompanied by many critical drawbacks:

1) The network becomes non-transparent to servers, and servers also cannot be independent of the network.
2) The Operating System kernel or the network protocol of a server has to be modified to implement the automatic fault-tolerant routing and flow-level load-balancing. Generally, the OS must be able to detect current network status (e.g., connectivity, variation of delay, etc.) and discover feasible alternative routing paths in real-time. Moreover, certain routing protocols (e.g., source routing scheme) must be implemented in order to allow a single flow transfer through multiple paths simultaneously. However, modifying the OS kernel could be very complicated and time-consuming in practice (e.g. DCell involves more than 13000 lines of C code [7]).
3) The server may become the bottleneck to the overall network performance and leads to additional packet delays, increased packet loss ratio and deceased aggregate bottleneck throughput, especially when suffering from insufficient software resources, for example CPU time and memory bandwidth [7], and servers cannot completely focus on computing and providing regular services.
4) A certain number of NICs are needed to be installed on each server to meet the future scaling out, which may be not very practical since most of current data center servers only have two built-in ports.
5) It is difficult to implement a green data center since servers cannot be powered off even when they are idle for computing because they still undertake forwarding tasks.

C. HYBRID NETWORK TOPOLOGY

In recent years, researchers have proposed several hybrid packet and circuit switched data center network architectures, and the typical representatives are c-Through [20] and Helios [37]. Fig.5 illustrates a electrical/optical hybrid network architecture. Compared with packet switching, the optical circuit switching can provide high bandwidth and low latency in transmission, and also consumes lower energy. However, optical circuit switching cannot achieve full bisection bandwidth at packet granularity. Furthermore, the optics also suffers from slow switching speed which can take as long as tens of milliseconds.

Take c-Through [20] for example, it augments the traditional hierarchy of packet switches with a high speed, low complexity, rack-to-rack optical circuit-switched network to supply high bandwidth to applications. In this structure, ToRs are connected via an electrical part and an optical part. The electrical part can be a switch or some switches forming a tree topology. The optical part is the focus of such structures and can dynamically change according to real traffic patterns. At any moment, it is a bipartite graph with direct links

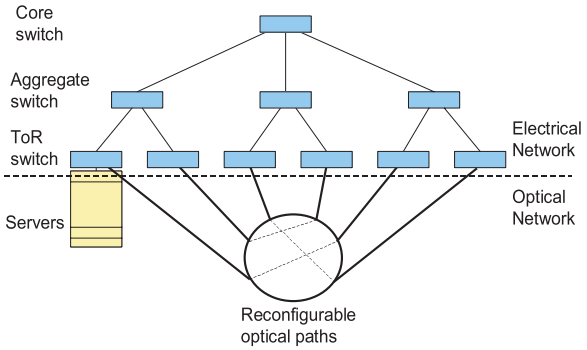


FIGURE 5. Hybrid network architecture [20].

assigned to heavy communication pairs via circuit. The routing in hybrid structures is straightforward: the optical part is a one-hop communication, and the electrical part is just a routing in a tree. Another novel proposal of hybrid data center network is Helios [37]. Helios is built in a form of 2-level multi-rooted tree consisting of pod switches and core switches. The end hosts and switch hardware need not to be modified. The decision to switch between circuit and packet is made transparently at the core layer depending on the dynamic communication patterns.

D. DIRECT NETWORK TOPOLOGY

Another audacious method proposed by researchers with great novelty is to directly connect servers to other servers. This is a switchless network interconnection without any switches, routers, or other network devices and associated cooling costs. The Torus-based architecture well implements the network *locality* forming the servers in close proximity of each other, which increases the communication efficiency. CamCube [2], NovaCube [45] and Small-World [29], which are built based on 3D-Torus (as shown in Fig.6 left), can be classified into this category. In this kind of network architectures, the servers use multiple NIC ports to get involved in the network infrastructure and also take a part in forwarding packets. The CamCube is designed targeting at shipping container-sized data centers. As shown in the right side in Fig.6, with the benefit of Torus architecture and the flexibility offered by CamCube API, it allows applications to implement their own routing protocols so as to achieve better application-leve
performance.

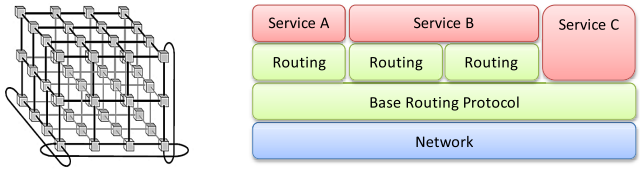


FIGURE 6. A 3D-Torus topology (left) and the CamCube internal structure (right).

TABLE 1. The comparison between some different network architectures.

	Fat Tree (3 layers)	VL2 (3 layers)	DCell (2 layers)	BCube (2 layers)	FlatNet (2 layers)	SprintNet (2 layers)
Servers Number	$\frac{n^3}{4}$	$\frac{(n-2)n^2}{4}$	$n(n+1)$	n^2	n^3	$(\frac{c}{c+1})^2 n^2 + \frac{c}{c+1} n$
Links Number	$\frac{3n^3}{4}$	$\frac{(n+2)n^2}{4}$	$\frac{3n(n+1)}{2}$	$2n^2$	$2n^3$	$\frac{c^2 n^2}{c+1} + cn$
per Server	3	$\frac{n+2}{n-2}$	$\frac{3}{2}$	2	2	≥ 2
Switches Number	$\frac{5n^2}{4}$	$\frac{n^2}{4} + \frac{3n}{2}$	$n+1$	$2n$	$2n^2$	$\frac{c^2}{c+1} n + c$
per Server	$\frac{5}{n}$	$\frac{n+6}{n^2-2n}$	$\frac{1}{n}$	$\frac{2}{n}$	$\frac{2}{n}$	$\frac{c+1}{n}$
Bisection Bandwidth	$\frac{n^3}{8}$	$\frac{n^2}{4}$	$\frac{n^2}{4} + \frac{n}{2}$	$\frac{n^2}{2}$	$\frac{n^3}{4}$	$\frac{c^2 n^2}{2(c+1)^2} + cn$
per Server	$\frac{1}{2}$	$\frac{1}{n-2}$	$\approx \frac{1}{4}$	$\frac{1}{2}$	$\frac{1}{4}$	$\frac{1}{2} + \frac{(2c+1)(c+1)}{2(cn+c+1)}$
Network Diameter	6	6	5	4	8	4

However, some other researchers found that this design consistently suffers from poor routing efficiency compared to other designs and this was mainly due to the relatively long routing paths in the 3D torus – $O(N^{1/3})$ hops, with N servers. In order to overcome this limitation, Shin Ji-Yong, et al. proposed Small-World [29], which provides an unorthodox random data center network topology. It is constructed based on some regular topologies (such as ring, torus or cube) with the addition of a large number of random links which can reduce the average path length and help achieve higher routing efficiency. The degree of each node is limited to six taking the realistic deployment and low cost into consideration. In addition to traditional routing method, Small-World also provides content routing coupled with geographical address assignment, which in turn efficiently implements key-value stores. However, its optimal shortest path routing suffers poor worst-case throughput and poor load balancing, though it achieves the minimum average routing path length. Comparatively, NovaCube [45] was proposed by adding a set of most beneficial jump-over links to connect the farthest node pairs in regular Torus topologies thus halved the network diameter. Besides, coupled with the probabilistic weighted oblivious routing algorithm NovaCube also greatly improved the throughput and bisection bandwidth.

The 3D Torus interconnection is attractive for a variety of reasons. Firstly, it results in lower infrastructure cost and energy cost because it does not need switches. Even the wiring is simpler than other methods such as Fat-Tree. Secondly, it enjoys better reliability and fault-tolerance. The traditional architecture is usually constructed with a large number of switches. If in case one of the network devices fails, it will greatly impacts on the network performance and system reliability. For example, if a ToR switch fails, the whole rack of servers will lost the connection with the servers in other racks. Thirdly, the architectural symmetry of Torus topology optimizes the scalability and granularity of Clusters. It allows systems to economically scale to tens of thousands of servers, which is well beyond the capacity of fat tree switches. Fourthly, it can provide high network performance, which has been proven in high-performance systems and supercomputers, such as IBM's Blue Gene/L [39], Cray Gemini [41] and Blue Gene/Q (5D Torus) [9].

These good features of the Torus network given above conclusively demonstrates its capability in constructing a cost-effective and high performance data center network. However, apart from various advantages of Torus, it also suffers some critical shortcomings, such as its relatively long average routing path length and low worst-case throughput.

E. COMPARISONS BETWEEN TOPOLOGIES

In this subsection, some typical features of different architectures are explored. The analysis of the structural properties of these architectures are summarized in Table 1, which also presents some comparison among these network architectures.

1) NETWORK DIAMETER

The diameter indicates the maximum shortest path length (denoted by the number of links) among all the server pairs. Compared with other architecture proposals as shown in Table 1, SprintNet, Fat-Tree and VL2 stand out because of its relatively low network diameter. Comparatively, the network diameters of other architectures are much higher, some of which (e.g. DCell, BCube) increase accordingly as the number of layers increases, as shown in Fig.7.

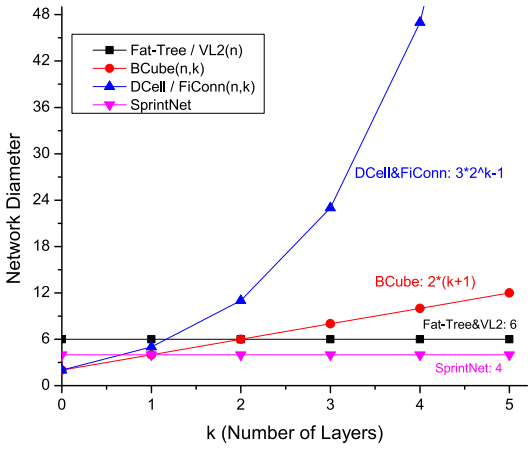


FIGURE 7. The comparison of network diameters among various topologies.

2) SCALABILITY AND PHYSICAL COST

For a 2-layer architecture, FlatNet achieves scalability of $O(n^3)$ which is the same as Fat-Tree, which is better than DCell, BCube, VL2 and SprintNet's $O(n^2)$. The wiring complexity of Fat-Tree is higher than DCell, BCube and SprintNet but lower than FlatNet. Additionally, 2-layer DCell and SprintNet use fewer switches in the network than other candidates and increases at the speed of $O(n)$. As for the cost of per server, the server-centric architectures (i.e. DCell, BCube, FlatNet, SprintNet) outweigh Fat-Tree and VL2.

3) BISECTION BANDWIDTH

The bisection bandwidth is depicted as the sum of link capacities between two equally-sized parts which the network is partitioned into. It can be used to measure the worst-case network capacity [19]. FlatNet and Fat-Tree achieve best bisection bandwidth at $O(n^3)$ level, and their bisection bandwidth per server is a constant, which is better than VL2.

IV. TRANSPORT PROTOCOLS IN DATA CENTER NETWORKS

Data centers are faced with higher requirements and more challenges in data transmission than the Internet, such as microsecond RTTs, decreased level of multiplexing, low packet loss ratio, and incast [48] problem which is caused by the synchronized workloads. However, the traditional TCP, which deals with congestion relying on packet drops having milliseconds feedback loops, cannot satisfy the high requirements of the data center network. As a result, many efficient schemes are proposed for data centers to address the issues of flow scheduling, congestion control, and reducing latencies. These works can be generally classified into two groups as below:

- *Deadline-Agnostic Protocols:* In these protocols they don't take the flow completion time (or latency) into consideration and only focus on the congestion control and flow scheduling, though they can reduce the latency to some extent by their efficient congestion control mechanisms. These kind of protocols include DCTCP [33], MPTCP [12], ICTCP [25], and RPS (Random Packet Spraying) [3].
- *Deadline-Aware Protocols:* This type of protocols not only aim to improve the network congestion, but also considers the flow completion time and latency by prioritizing near-deadline flows over far-deadline flows in case of congestion. The works like D^{2}TCP [6], D^3 [8], DeTail [17], pFabric [36], PDQ (Preemptive Distributed Quick) [10], HULL (High-bandwidth Ultra-Low Latency) [35] can be classified into this category.

In the following subsections, some typical representative proposals are provided with detailed analysis and discussions.

A. DEADLINE-AGNOSTIC PROTOCOLS

1) DCTCP

Relying on Explicit Congestion Notification (ECN), DCTCP (Data Center TCP) [33] as a new variant of TCP is specially designed for data center networks. DCTCP is motivated by the careful observation on the traffic measurements in a real production data center cluster that short flows require low latency and high burst tolerance while long flows need high utilization with high throughput. The switches used in DCTCP are all swallow buffered.

The main idea of DCTCP is very simple, and it is described as below:

At the switch side DCTCP applies an active queue management approach, which actively mark the packet with the CE codepoint if the queue occupancy exceeds the threshold K. Then the receiver ACKs every packet, which is marked with CE codepoint, with ECN-Echo flag. Finally, the sender adjusts its congestion window size in proportion to the fraction of packets that are marked (denoted as α). The window size is updated according to the following formulas:

$$\alpha = (1 - g) * \alpha + g * F \tag{1}$$

$$cwnd = cwnd * (1 - \frac{\alpha}{2}) \tag{2}$$

where α indicates the fraction of marked packets, which implies the congestion level. F is the fraction of marked packets in the previous window of data. g provides a weight to the new sampled packets against the past. $cwnd$ denotes the window size. It can be seen that a larger α results in a smaller window size $cwnd$, and the window size is adjusted in proportion to the extent of congestion other than being cut by a factor of 2.

The experiment shows that DCTCP can achieve high throughput while maintaining low buffer occupancies. However, DCTCP cannot handle the incast problem when the number of flows is large, and the deterministic marking mechanism may consecutively mark the packets from the same input port which then causes the TCP Outcast problem. Besides, DCTCP is a deadline-agnostic protocol that treats all flows equally in case of congestion, regardless of whether their deadlines are near or far. This can result in as high as 7% of deadlines to be missed and 25% missed deadlines at a high fan-in degree [6], [8]. Moreover, DCTCP is a single-path data transfer scheme without utilizing the path diversity in data center networks, which may lead to imbalanced link utilization and limit the achievable throughput. Additionally, how to define the best value of threshold K is still a pending issue.

2) MPTCP

The data center network is usually richly interconnected with many equal cost multiple paths (as shown in Fig.1, Fig.2, Fig.3, Fig.6). For example, in the Fat-Tree network architecture with a 1:1 bandwidth oversubscription ratio, there are $\frac{k^2}{4}$ equal cost paths between any pair of inter-pod servers, where

k is the number of ports per switch. However, the researchers observed that the traffic matrix that should be able to fill the network can not fully utilize the network with relevant unfairness and unexpected low throughput when flows traverse only a single path. Based on this careful observation, Costin Raiciu et al. proposed a multipath routing scheme which leverages the existence of multiple paths at the aggregation layer and the core layer in data centers to effectively utilize available bandwidth and achieve higher throughput.

The main idea of MPTCP is to divide each source-destination flow into number of subflows and relies on ECMP routing to hash them to different paths so as to avoid congestion and achieve better utilization and throughput. The subflows can use the same source IP and destination IP as the first flow, but use different ports to denote themselves as new flows. If the host has additional IP addresses, the subflows also can use them as their flow identifier. Each subflow has its own congestion window and sequence space, but the window size is adjusted based on the extent of congestion of its path and depending on the total window size, which can greatly contribute to achieve better load balancing and avoid congestion. For example, the subflows on congested paths have smaller window size and increase more slowly than the subflows on less congested paths with larger window size. In this case, the traffic workload will be moved to the less congested paths.

As summarized in this paper, MPTCP achieves three main benefits:

- It improves the network utilization by exploiting the path diversity and good congestion control, which results in a better aggregate throughput.
- By splitting each flow and distributing them on different paths, the flows with different MPTCP connections can achieve similar throughputs and the congestion on the network core can be more evenly distributed, which reflects the fairness.
- It achieves better reliability in data transmission. MPTCP can simply route around the failure node (link/switch) and move the traffic to different available paths without interruption.

MCTCP seems to be very attractive and promising for real deployment, and the experimental results are very convincing as well. However, MPTCP has no evaluation on real workloads and more practical evaluations are required to be conducted. Besides, multipath routing cloud worsen the TCP incast issue.

3) ICTCP

TCP incast congestion usually happens when a parent server simultaneously places a quest to a cluster of worker server for data, and after computation all worker servers will respond to the parent server with requested information. The phenomenon that many worker servers simultaneously send data to a same parent server will result in egress congestion in the last hop switch connected to the parent server, exceeding the egress port buffer accompanied with lots of packet drops. The worker servers detect the lost packet on the basis of missing ACKs and resend data after RTO (around 200ms) incurring the slow start period of TCP and heavily degrading the throughput with increased latency. A typical incast scenario is illustrated in Fig.8.

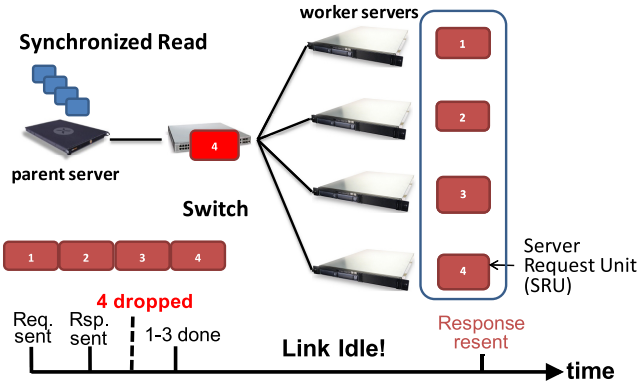


FIGURE 8. The scenario of TCP incast.

TCP incast is a common problem in cloud data centers where implementing many distributed storage and computing frameworks such as MapReduce [18], Hadoop [47] which produce lots of barrier synchronized many-to-one communication patterns. Against this issue, Microsoft Asia researchers proposed ICTCP (Incast congestion Control for TCP), which is a specialized TCP variation dedicated to mitigating the incast problem. Different from other approaches which focus on exploiting a fine grained timeout value [44] or controlling switch buffer occupation to avoid overflow (e.g. DCTCP), ICTCP takes a flow control approach at the receiver side to avoid packet loss before incast happens other than recovering after loss. The intuition of controlling incast at receiver side is that the receiver always knows the achieved throughput already and the available bandwidth left. ICTCP is proposed based on three observations: firstly the receiver uses the available bandwidth as the signal to prevent potential incast congestion; secondly, the interval of congestion control is conducted according to per-flow RTT independently; thirdly, both congestion state and application requirement should be taken into consideration for the adjustment of window size.

Briefly, the key idea of ICTCP is to adjust the receive window of each connection by estimating the available bandwidth and RTT. All low-RTT (less than 2ms) TCP connections should be jointly considered to adjust the receive window at each interface of the receiver. Before increasing its receive window, each flow should check if its estimated potential throughput exceeds the available bandwidth BW_A, which is computed as:

$$BW_A = max(0, \alpha * C - BW_T) \tag{3}$$

where C is the link capacity, BW_T indicates the already used bandwidth of total traffic, and α provides a safety margin. The receive window of each connection is adjusted based on measured throughput b_i^m and expected throughput b_e^m, which

are calculated as below:

$$b^m_{i,new} = max(b^s_i, \beta * b^m_{i,old} + (1-\beta) * b^s_i) \quad (4)$$

$$b^e = max(b^m, rwnd_i/RTT_i) \quad (5)$$

$$d^b_i = (b^e_i - b^m_i)/b^e_i \quad (6)$$

where b^s_i is calculated as total received bytes during the interval divided by RTT_i, $rwnd_i$ denote the window size, and $0 \leq d^b_i \leq 1$. In their experiments, the receive window is increased when $d^b_i \leq 0.1$ and there is enough available bandwidth, while decreased by one MSS if $d^b_i > 0.5$, otherwise the window keep the same. Fig.9 depicts a systematically designed ICTCP.

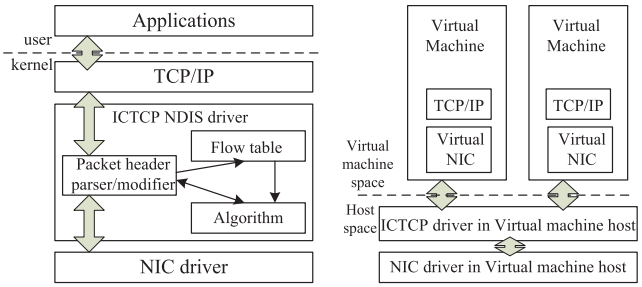


FIGURE 9. Modules in ICTCP driver and software stack for VM support [25].

The experimental results show that ICTCP avoids congestion effectively, and achieves almost zero timeout for TCP incast. However, it should integrate ICTCP into TCP stack, which needs change the TCP source codes. Alternatively, they have developed ICTCP as a NDIS driver on Windows OS kernel, which is complicated to be implemented and not compatible with other operating systems. Besides, ICTCP works under the assumption that the incast happens at the last hop and the bottleneck link is directly connected to the receiver, then the receiver estimate the available bandwidth and RTT to compute receive window. However, it is very difficult to estimate exact available bandwidth and RTT in real-time, and ICTCP also will fail to work well if the bottleneck link is not directly connected to the receiver.

B. DEADLINE-AWARE PROTOCOLS

1) D^3

D^3 (Deadline-Driven Delivery) [8] is a deadline-aware control protocol which is specially customized for data center networks. D^3 is a flow level congestion control protocol, where deadlines are provided by the applications and associated with flows other than packets. The goal of D^3 is to maximize application throughput (meet as many deadlines as possible), better accommodate flow bursts (burst tolerance), and achieve high utilization.

D^3 assumes that applications can provide the information of flow size and deadline at flow initiation time. Based on this information, the end host can request the desired sending rate for the flow with the rate $r = \frac{s}{d}$. This desired rate is brought in the packet header traversing the routing path, and routers running D^3 tries to allocate a proper rate. A set of allocated rate by the routers along the path will be fed back to the source host, and then the source sends data using the minimum of the allocated rate for next RTT. The source will periodically request a new rate allocation based on the deadline and remaining flow size.

As for the rate allocation, the routers apply a greedy approach to allocate rates for each flow. Primarily, a router tries to allocate each deadline flow with its requested rate r. After satisfying all the deadline flows, if there is still available bandwidth, then router distributes the available bandwidth amongst all flows including both deadline and non-deadline flows. Therefore, the final allocated rate a to each flow is: $a = r + fs$ for deadline flow and $a = fs$ for non-deadline flows, where fs is the fair share of the available bandwidth. There is another case, which is there is no enough bandwidth for allocation to all flow with their desired rate. At this time, the router running D^3 will apply a greedy algorithm to satisfy as many deadline flows as possible, and then assign a base rate (header-only packet per RTT) to the remaining deadline and non-deadline flows.

There are some practical shortcomings in D^3. Firstly, D^3 needs modifications to switch hardware to deal with the requests, which is not practical for deployment. Secondly, D^3 cannot coexist with legacy TCP, where the priority based bandwidth allocation mechanism running on routers is not compatible with the legacy TCP flows without the information of deadline, flow size and desired rate. Thirdly, the greedy bandwidth allocation approach may allocate bandwidth to far-deadline flows prior to near-deadline flows if the far-deadline flow arrive earlier than the near-deadline flow, which may increase missed deadlines. Fourthly, the periodical rate requests sent from sources will bring some overhead.

2) D^{2}TCP

D^{2}TCP [6] can be regarded as an improved extended version of DCTCP. The major improvement is that D^{2}TCP takes deadline into consideration when resizing the congestion window for each flow apart from using ECN flag. Therefore, D^{2}TCP is a deadline-aware protocol. Different from D^3's centralized bandwidth allocation at the switches, D^{2}TCP inherits the distributed and reactive nature of TCP without any changes to the switch hardware. D^{2}TCP adjusts the congestion window *cwnd* follows the rules as below:

$$\alpha = (1-g) * \alpha + g * F \quad (7)$$

$$p = \alpha^d \quad (8)$$

$$cwnd = cwnd * (1 - \frac{p}{2}), \quad if\ p > 0$$
$$= cwnd + 1, \quad if\ p = 0 \quad (9)$$

where α, g, F, $cwnd$ have the same meanings as in Eq.(1). The factor d reflects the extent of deadline urgency, where a larger d suggests a closer deadline. The penalty p is computed based on a and d.

Strictly following the above rules, D^{2}TCP will throttle the far-deadline flows aggressively and the near-deadline

flows will not back-off or only a little. Because D^{2}TCP is built upon DCTCP, so it also has high burst tolerance. Besides, according the experiment results, D^{2}TCP performs much better than DCTCP and D3 in terms of reducing the ratio of missed deadlines. However, there is a big concern about the scalability in terms of number of worker nodes in parallel. For example, if the number of worker nodes increases to a certain number, then it will fail to avoid TCP Incast. Additionally, more experiments and evaluations are required to verify whether D^{2}TCP can solve TCP Outcast problem. Furthermore, D^{2}TCP cannot deal with the following three special cases:

- If most of flows are near deadline flows, then they will not back off or back off little, and this may still cause congestion.
- If most of flows are far deadline flows, then they will back off greatly, and this will result in low throughput and low network utilization.
- In case all flows are near deadline flows, and competing the bandwidth, if no flow is backed off then all flows will miss their deadlines, if one or some could volunteer to back off, then all the rest can be finished before their deadlines. The problem is how to determine which flow(s) should be sacrificed in a proper way.

3) DeTail

DeTail [17], which was proposed by David Zats et al., is a cross-layer network stack (see Fig.10) and also a multipath-aware congestion management scheme. It is specially designed targeting at reducing the long tail of flow completion times in data center networks. Detail achieves this goal by coordinating cross layers to cooperate together to prevent congestion at lower layers and do adaptive load balancing at upper layer.

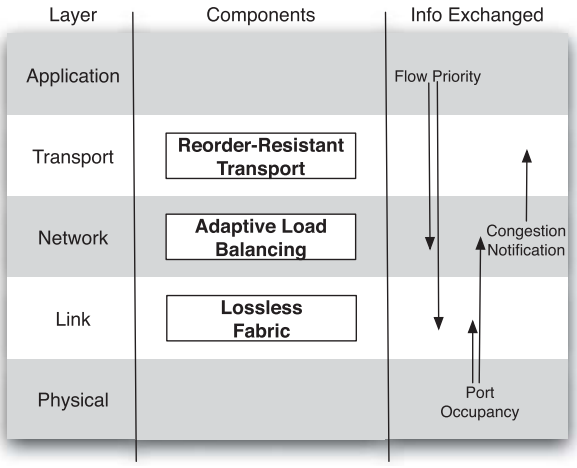


FIGURE 10. The Detail network stack uses cross-layer information [17].

All flows in DeTail are assigned with priorities (up to eight priorities) and switches perform strict priority queuing at both ingress and egress port queues. The labor division of each layer in DeTail can be generally summarized as follows:

- *Link Layer:* Flow control is conducted in this layer to achieve a lossless fabric. Based on the ingress queue occupancy, Detail uses Pause/Unpause message to implement a hop-by-hop flow control. In order to mitigate HOL (head-of-line) blocking, Details adopts Priority Flow Control mechanism cooperated with adaptive load balancing at network layer and Explicit Congestion Notification at transport layer.
- *Network Layer:* In this layer, DeTail implements packet-level adaptive load balancing according to the congestion level of each egress port (judged by the egress queue occupancies) by exploiting the path diversities in data center networks. This allows the packets always choose the lightly loaded path to transmit.
- *Transport Layer:* DeTails employs a modified TCP NewReno which disables fast recovery and fast retransmit, because there is already no packet loss due to the work in link layer. Besides, DeTail also uses ECN to mark the low priority flows (deadline-insensitive flows) when the drain bytes counter exceeds a threshold so as to prevent persistent congestion.
- *Application Layer:* The responsibility of this layer is mainly to set each flow with the one reasonable priority depending on their latency-sensitiveness.

According to their experimental results, the concerted close cooperation of different layers can achieve a reduction of flow completion times by 50% at 99.9th percentile. However, as noted in [16], it does not address network virtualization overlays. Besides, it does not consider the average flow completion time and deadline missing flows as well.

4) PDQ

PDQ (Preemptive Distributed Quick) [10], which was proposed by Chi-Yao Hong et al., is also a protocol devoted to reducing flow completion time and meeting flow deadlines. Similar to previous works like D^3, PDQ also explicitly assigns rates to individual flows at switches to meet flow deadlines, while the difference is that PDQ proactively and preemptively assign rates based on the criticality of flows other than D^3's "first-come first-serve" manner. PDQ employs two real-time scheduling techniques to process the queues and schedule flows at switches, one is Earliest Deadline First (EDF) which aims to minimize missing deadlines, and the other is Shortest Job First (SJF) which targets at reducing the average flow completion time. Then PDQ implements the two centralized scheduling algorithms EDF and SJF in a totally distributed manner, which can avoid the required global knowledge of flow information (which is unrealistic in data centers), single point of failure, and significant flow initialization overhead. In order to implement the distributed scheduling algorithm, every packet sending from any senders will be attached with a scheduling header, which contains several flow-related state variables including current sending rate, flow deadline, its expected transmission time, the measured RTT, probing

frequency for paused flows, and so on. This flow-related information will be shared with switches on the path and switches assign a sending rate for each flow by modifying the scheduling header of packets. When sender receives the acknowledgement, it will send data with the assign rate or to pause. Besides, the authors also proposed a multipath PDQ (M-PDQ) aiming to improve network reliability and network utilization by enabling switches to support flow level Equal-Cost Multipath (ECMP). The overview of PDQ's particular design on senders, switches, and receivers is provided as below:

- *Sender Side:* After receiving the ACK for flow initialization SYN packet, the sender sends data at the rate assigned by switches or to pause, where the information is encoded in the scheduling header of ACK packet. As mentioned above, the scheduling header is carried with every sending packet. Whenever an ACK packet returns, the sender updates each flow's sending rate according to the rate information enclosed in the scheduling header of the ACK packet. Once the flow is paused (i.e. the rate is zero), the sender complies *Suppressed Probing* mechanism to send a probing packet with only a scheduling header for the purpose of collecting rate information from switches and the receiver. Finally, the sender uses a heuristic *Early Termination* to terminate a flow by sending a TERM packet whenever it cannot finish before its deadline.
- *Switch Side:* In each of PDQ switch, it mains the states of $2k$ (k denotes the number of in-flight flows) most recent and most critical flows, so that it can immediately unpause one paused flow when a flow finishes. The bandwidth is assigned at switches to flows using EDF and SJF scheduling principles, where EDF has a higher priority. A PDQ switch is also responsible for computing the flow sending rate feedback, which is encoded in the scheduling header of every packet, by a specially designed flow controller and rate controller. In addition, PDQ cooperates with an RCP [38] rate controller and uses Early Start to deal with two different cases of under-utilization. The final goal of PDQ switches is to complete the most critical flows with highest priorities as quickly as possible.

  In particular, the flow scheduler is tasked to control sending or pausing one specific flow by checking if it is paused by any other switches. This design can be further optimized by applying *Early Start* scheme (which provides a seamless flow switching to address the low-utilization issue), *Dampening* scheme (which deals with the temporary instability in the switch state), and *Suppressed Probing* scheme (which aims to reduce the significant bandwidth overhead caused by frequent probing packets).

  The rate controller's main responsibility is to control the aggregated flow sending rate based on a variable C, which is maintained by the rate controller and updated every 2 RTTs. PDQ benefits a lot from the rate controller in tolerating the congestion caused by the packet lost (which contains pausing information to inform a sender to stop sending flows) and mitigating the effects brought by the use of *Early Start*.
- *Receiver Side:* The key modification to a PDQ receiver is that it needs to copy the scheduling header of the data packet, which has been tagged by the switches along the path, to its ACK packet, and adjust the rate feedback field if necessary.

The experimental results have convinces a better performance of PDQ in both flow completion time and application throughput compared with other schemes such as D^3, TCP, and RCP. However, it is quite complicated and difficult to implement in practice. Besides, the switches of PDQ needs to maintain a set of flow state variables for a number of flows, which brings a certain degree of burden to switches. Additionally, the frequent probing packet and the scheduling header attached to every packet also increase the network overhead to some extent. Moreover, in order to obtain the sending rate, each flow starts with a SYN to collect the rate feedback and terminates with a FIN to release the bandwidth, this certainly results in an extra RTT latency on every flow, which is a quite serious issue since over 50% of flows in data centers are less than 1KB [5] which can be finished in just one RTT.

5) pFabric

As claimed, pFabric [36] is a near optimal minimalistic data center transport design proposed by M. Alizadeh et al. recently. pFabric provides a clean-slate transport design, which is simple but efficient. The key idea of this pFabric is to reduce flow completion times for short flows and minimize average flow completion time for long flows by decoupling flow scheduling and rate control. Specifically, as shown in Fig.11, the switches in the fabric takes the responsibility of flow scheduling and the rate control is implemented at the side of end hosts. The simply designed switch-based and priority-based flow scheduling mechanism in turn simplifies the rate control. Unlike DeTail which is a lossless fabric, pFabric implements its rate control based on a signal of high persistent packet loss rates. Briefly, the design of pFabric can be summarized into two phases as below:

1) *Switch Design (Priority Scheduling/Dropping):* Firstly, each switch has very small buffers, whose size is

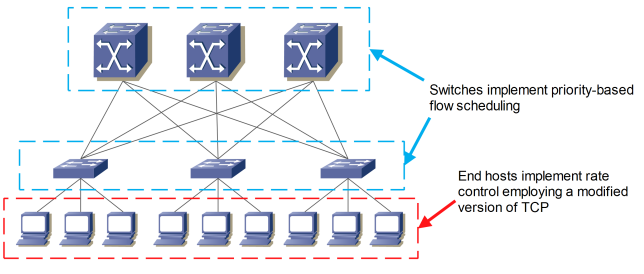


FIGURE 11. Decouple flow scheduling from rate control in pFabric.

around $2 \times$ BDP per port (e.g. if link capacity is 10Gbps and RTT $= 15\mu$, then buffer $=$ 30KB). Secondly, each packet of a flow is encoded with a priority, which represents remaining flow size or deadline depending on the scheduling objective. Thirdly, a priority queuing is performed at each switch, where the packets with higher priorities are firstly transmitted. Fourthly, when a new packet arrives to a port and its buffer is full, if the priority of this new packet is not larger than the lowest priority buffered packets, then it is dropped. Otherwise, the new packet replaces the lowest priority packet, which is dropped directly.

2) *Rate Control Design:* The main task for rate control at end hosts is to prevent the congestion collapse when elephant flows collide. Besides, the end hosts are also responsible for assigning a priority to every packet. End hosts employ a modified version of TCP to conduct the rate control. The main modification to TCP is that pFabric throws away dupACKs based fast retransmission, and uses a fixed RTO ($3 \times$ RTT used in their experiments) for detection of packet drops. Initially, flows start at line rate with one BDP sized window (12 packets in their experiments). Similar to traditional TCP, flows will enter into slow start once timeout happens and execute additive increase for every packet SACK. If there is a persistent packet loss (detected by timeout) which implies a congestion collapse, it will be forced into probe mode, and then come back to slow start period until receiving the acknowledgement of probing packet.

Compared with other schemes such as PDQ and DCTCP, pFabric achieves a significantly better performance in reducing flow completion time. The simulation results exhibit that the mean flow completion time achieved by pFabric is already very close to that of ideal flow scheduling scheme for both web search workload and data mining workload. However, the performance of pFabric in achieved application throughput is not as good as PDQ, probably because of its quite small buffers on switches. Besides, pFabric requires new switches, which can implement their flow scheduling mechanisms, and some changes to end hosts. Additionally, there are still many challenges for incremental deployment with existing schemes, for example, how to decide the number of priority queues, how to determine thresholds for each priority, how to efficiently measure the dynamic flow size distribution, and so on.

To sum up, based on the results of extensive experiments, with respect to the flow completion time we can derive that pFabric > PDQ > DCTCP > TCP-DropTail.[1]

V. NETWORK RESOURCE SHARING IN DATA CENTERS

As highly multiplexed shared environments, cloud data centers are equipped with a large number of physical servers and virtual machines (VM) hosted in servers to simultaneously offer multiple tenants with on-demand use of computing resources in a pay-as-you-go manner. As noted in [28] the network bandwidth in data centers is a scare resource, thus how to efficiently and securely share the network resource among multiple tenants is a key concern. However, the network resource is far more difficult to share than physical resources like CPU and memory. The current widely used TCP-based resource allocation and congestion control can not provide robust performance isolation, and face with denial of service (DoS) attacks and performance interference. In response to this issue, many research works have been motivated and conducted. The unique goal of these works is to achieve a bandwidth guaranteed data center network providing network performance isolation for multiple co-existed untrusted tenants following some fair sharing policies in a virtualized environment, so that the cloud data center can (i) be scalable to dynamic changing number of VMs and tenants, (ii) provide predictable network performance for users, (iii) be robust to malicious/untrusted tenants, and (iv) achieve service level flexibility (i.e., minimum and maximum performance guarantee). There are some great proposals that achieve specific network sharing policies, among which the most typical research representatives include SecondNet [11], Oktopus [27], Gatekeeper [23], Seawall [4], NetShare [43], FairCloud [31].

A. SecondNet

SecondNet provides a static allocation mechanism to achieve bandwidth guarantee in a virtualized data center network. It regards virtual data center (VDC) as the unit of resource allocation, where a VDC is defined as a set of VMs with a customer-supplied IP address range and an associated service level agreement (SLA) based on VM-to-VM pair network requirements. Tenants are mapped into different VDCs, and each VDC is assigned its own private IP address space to address isolation. VMs within the same VDC can communicate via layer-2 Ethernet and the inter-VDC communications must go through layer-3 gateways. Besides, SecondNet provides a service model of three VDC types to achieve different level of bandwidth guarantee based on different traffic patterns. In order to achieve an efficient and low time-complexity VDC allocation, SecondNet uses a logically centralized VDC manager, which manages all resources and fulfills resource allocations. All the states of the virtualization and bandwidth reservation are stored in server hypervisors and the switches in the network are stateless, which results in high scalability. The specially designed port-switching based source routing (PSSR) using port predefined port numbers instead of MAC addresses makes SecondNet applicable for all network topologies and be easily deployed with current commodity switches. However, the static bandwidth reservation scheme probably can leave the network underutilized because the available bandwidth cannot be shared between tenants. Besides, to some extent the performance of SecondNet depends on the structure of network topology, where

[1]TCP-DropTail is a standard TCP-New Reno with SACK and drop tail queues.

different topologies may achieve different performance. Additionally, SecondNet only counts the intra-datacenter bandwidth in allocation but without taking the Internet-access bandwidth, for that running web applications, into consideration.

B. OKTOPUS

Oktopus proposes a static per-tenant reservation throughout the network for the sake of achieving bandwidth guarantee and performance guarantee. Oktopus is initially proposed based on a careful observation that: The multi-tenant cloud data centers provide tenants with computing resources according to SLA, but due to the scare network resource and inefficient sharing mechanism the data centers usually cannot guarantee the performance and meet the requirements. The application's performance unpredictability induced by the varying network performance ultimately brings great negative impacts on both provider revenue and customer's satisfaction. Against this issue, Oktopus designs two novel virtual network abstractions, Virtual Cluster (VC) and Virtual Oversubscribed Cluster (VOC), to enable a symbiotic tenant-provider relationship, where tenants get predictable performance at a lower cost and providers increase their revenues. The VMs allocated into VC receive a bandwidth guarantee while VOC allocates VMs with one additional parameter of the oversubscription ratio. Oktopus contains two main components: management plane and data plane. Management plane is responsible for allocating tenant requests to physical infrastructure meeting the tenant network bandwidth requirements by employing a logically centralized network manager. Data plane achieves the enforcement of tenant bandwidth requirements through rate limiting at end-host hypervisors, irrespective of traffic characteristics (like type of protocols, number of flows between VMs, etc.). Moreover, Oktopus designs a greedy virtual cluster allocation algorithm for the requests allocation. As claimed by the authors, Oktopus can be incrementally deployed today, but only limited to tree-based topologies. Besides, dynamical changes of requirements from multiple tenants can result in chain reactions in VOC. Another problem is that Oktopus's greedy allocation algorithm may allocate unavailable resources to tenants if it misses any information of resources resided several layers lower than the current layer. In addition, like SecondNet, Oktopus also fails in considering the Internet-access bandwidth when doing bandwidth allocation.

C. GATEKEEPER

The authors summarized the key properties of a practical solution to network performance isolation, which should be scalable to dynamic changing number of VMs and tenants, predictable network performance for users, robust to malicious tenants, service level flexibility in terms of the minimum and maximum performance guarantee. Aiming to meet these requirements, the authors designed Gatekeeper, an I/O virtualization control system, which can achieve network performance isolation for mutually untrusted tenants in a shared virtualized data center. Gatekeeper provides a per-VM based fairness and per-vNIC (virtual NIC) link bandwidth guarantees for both ingress and egress traffic. Gatekeeper defines both minimum bandwidth guaranteed rate and maximum rate for scheduling bandwidth to each VM pair. The minimum bandwidth guaranteed rate is achieved by a traditional weighted fair scheduler. When both sender and receiver have extra bandwidth, each vNIC can exceed its guaranteed allocation to fully utilize the network. Each vNIC interface is associated with a set of rate limiters and counters. The counters are created (when receiving packets from new senders) and deleted (after timeouts) dynamically, storing the information about the remote vNICs of senders. Each receiving vSwitch periodically measures its traffic rates and generates congestion feedback messages if the aggregate rate exceeds 95% of the physical link capacity or one of its vNIC exceeds its maximum rate. The congestion message will be generated for the vNIC that exceeds its minimum guarantee most, and the receive rate of this vNIC is reset to its minimum guaranteed rate, then inform its senders with a explicit computed rate. Afterwards, the sender uses rate limiter to reduce its sending rate based on this congestion message. The authors have developed an incomplete version of Gatekeeper prototype for their initial experiments, where its function of dynamic creating and deleting rate limiters still remains to be implemented.

D. SEAWALL

Seawall implements an efficient resource allocation scheme by congestion controlled VM tunneling, which achieves VM-pair based fairness, enforces isolation and provides a simple and flexible service interface for tenants. In Seawall, every VM is associated with a weight, and the bandwidth sharing is proportional to weight and is agnostic to traffic patterns and transport protocols. Seawall consists of one shim layer on the forwarding paths at sender and receiver, the hypervisor rate limiter and bandwidth allocator. The traffic is sent through congestion-controlled tunnel per VM ⟨source, destination⟩. Seawall adopts a link-oriented congestion control accepting weight parameter. For every source VM, it runs a separate distributed control loop instance for every active link to generate per-link rate limit, and afterwards converts it to per-tunnel rate limits, and periodically collects link-level congestion feedback (e.g. ECN marked, packet loss ratio). This mechanism can prevent harmful behaviors of selfish or malicious tenants. Briefly, Seawall employs hypervisor rate limiters and end-to-end rate controller to achieve performance isolation while providing good aggregate throughput and efficient network utilization. However, there is potential unfairness when VMs of different tenants on the same server receiving traffic from VMs on other servers. More specifically, because Seawall allocate link bandwidth among the total number of communicating VMs through the link, so if there are two VMs on the same server belonging to two different tenants and one VM receives traffic from many VMs on other servers while

TABLE 2. The comparison between several schemes.

Scheme	Scalability	Load Balancing	Reliability	Bandwidth Guarantee	Quality of Service
SecondNet	Good	No	Yes	Yes	Good
Oktopus	Good	No	Yes	Yes	Good
Gatekeeper	Good	No	Yes	Yes	Good
Seawall	Good	No	No	No	Good
NetShare	Poor	Yes	No	No	Good

one VM receives traffic from just one sender, then the former tenant will be allocated with higher server link bandwidth than the latter one.

E. NetShare

Different from SecondNet and Seawall which provides bandwidth guarantees for VM pairs, NetShare proposes a mechanism to share the link bandwidth taking a per-tenant network-wide perspective, which is similar to Oktopus. The services are specified with different weights, and NetShare provides a service-level weighted hierarchical max-min fair approach for bandwidth allocation among services other than individual connections. NetShare can be possibly implemented by adopting three mechanisms: group allocation leveraging TCP, rate throttling for UDP and centralized bandwidth allocation. Group allocation relies on TCP and together with fair queuing to achieve max-min fair bandwidth allocation among services. In their implementation, they use deficit round robin (DRR) [34] to configure fair queuing and ToS bits of the header to differentiate services. Rate throttling is used to avoid the excessive bandwidth consumption of malicious applications and aggressive UDP flows. This is achieved by the rate throttling shim layer, which is placed under UDP in each host. The shim layer takes responsibility of measuring the received traffic and adjusting its sending rate by collaborating with its corresponding communicators. Finally, the centralized bandwidth allocator provides advanced allocation policies in addition to max-min fairness. It consists of four steps: rate measurement for each service, reporting predicted rates to centralized allocator, centralized rate calculation for flows and services, rate enforcement using token bucket rate-limiters. Multipath is enabled in NetShare with the use of ECMP, and NetShare requires no hardware changes to switches or routers. However, it does not strictly provide minimum bandwidth guarantees to services. Besides, relying on a centralized controller results in some issues such as scalability, timely responsiveness, great burden for heavy workload with high rates, and so on.

F. FairCloud

FairCloud claims that the network sharing policy should meet three key requirements, which can be briefly summarized as minimum bandwidth guarantee, maximum network utilization, and network proportionality (the shared resources should proportional to their payments). However, not all goals are achievable simultaneously and a fundamental tradeoff should be made when sharing cloud networks. In the process of going from goals to solutions, FairCloud takes an intermediary step and breaks down goals into lower level, a set of simpler properties (work conservation, utilization incentives, communication pattern independence, symmetry) that are required by the goals. Then FairCloud proposes three allocation policies (PS-L at link level, PS-N at network level, and PS-P on proximate links) to guide sharing the network sharing with different tradeoffs on the aforementioned properties and goals. FairCloud can be regarded as the pioneer work to explore the tradeoff space among the intended network sharing properties.

G. COMPARISON

Table 2 gives a general comparison between some typical network sharing schemes in terms of scalability, load balancing, reliability, bandwidth guarantee and QoS.

VI. CONCLUSION

Data centers are playing a more and more important role in the new era of could computing, inevitably accompanied with more and more challenges and issues in various aspects. In this survey paper, we provided a comprehensive review on the related research literature on data centers. We discussed many novel proposed schemes and techniques in different research areas ranging from DCN topologies, transport protocols, to network sharing mechanisms in data centers. Although these proposals address many issues and improve the network performance, there are still challenging and critical issues that are left for researchers to handle. Nevertheless, a good understanding of the data center networks would enable us design a more efficient, cost-effective, reliable and sustainable data center.

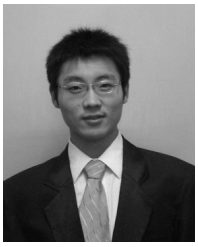
ACKNOWLEDGMENT

Besides, the authors would like to express their thanks and gratitudes to the anonymous reviewers for their time in reviewing this paper.

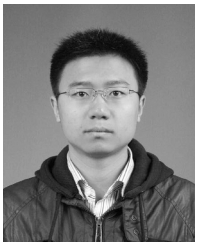
REFERENCES

[1] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," in *Proc. ACM SIGCOMM*, 2008, pp. 63–74.
[2] H. Abu-Libdeh, P. Costa, A. Rowstron, G. O'Shea, and A. Donnelly, "Symbiotic routing in future data centers," in *Proc. ACM SIGCOMM*, 2010, pp. 51–62.
[3] A. Dixit, P. Prakash, Y. C. Hu, and R. R. Kompella, "On the impact of packet spraying in data center networks," in *Proc. INFOCOM*, Apr. 2013, pp. 2130–2138.
[4] A. Shieh, S. Kandula, A. Greenberg, C. Kim, and B. Saha, "Sharing the data center network," in *Proc. 8th USENIX Conf. Netw. Syst. Design Implement. (NSDI)*, 2011, p. 23.

[5] A. Greenberg *et al.*, "VL2: A scalable and flexible data center network," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 39, no. 4, pp. 51–62, Oct. 2009.
[6] B. Vamanan, J. Hasan, and T. N. Vijaykumar, "Deadline-aware datacenter TCP (D2TCP)," in *Proc. ACM SIGCOMM*, 2012, pp. 115–126.
[7] C. Guo, H. Wu, K. Tan, L. Shi, Y. Zhang, and S. Lu, "DCell: A scalable and fault-tolerant network structure for data centers," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 4, pp. 75–86, Oct. 2008.
[8] C. Wilson, H. Ballani, T. Karagiannis, and A. Rowtron, "Better never than late: Meeting deadlines in datacenter networks," in *Proc. ACM SIGCOMM*, 2011, pp. 50–61.
[9] D. Chen *et al.*, "The IBM blue Gene/Q interconnection fabric," *IEEE Micro*, vol. 32, no. 1, pp. 32–43, Jan./Feb. 2012.
[10] C.-Y. Hong, M. Caesar, and P. B. Godfrey, "Finishing flows quickly with preemptive scheduling," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 42, no. 4, pp. 127–138, 2012.
[11] C. Guo *et al.*, "SecondNet: A data center network virtualization architecture with bandwidth guarantees," in *Proc. 6th Int. Conf. Co-NEXT*, 2010, p. 15.
[12] C. Raiciu, S. Barre, C. Pluntke, A. Greenhalgh, D. Wischik, and M. Handley, "Improving datacenter performance and robustness with multipath TCP," in *Proc. ACM SIGCOMM*, 2011, pp. 266–277.
[13] D. Li, C. Guo, H. Wu, K. Tan, Y. Zhang, and S. Lu, "FiConn: Using backup port for server interconnection in data centers," in *Proc. IEEE INFOCOM*, Apr. 2009, pp. 2276–2285.
[14] D. Lin, Y. Liu, M. Hamdi, and J. Muppala, "FlatNet: Towards a flatter data center network," in *Proc. IEEE GLOBECOM*, Dec. 2012, pp. 2499–2504.
[15] D. Lin, Y. Liu, M. Hamdi, and J. Muppala, "Hyper-BCube: A scalable data center network," in *Proc. IEEE ICC*, Jun. 2012, pp. 2918–2923.
[16] D. Crisan, R. Birke, G. Cressier, C. Minkenberg, and M. Gusat, "Got loss? Get zOVN!" in *Proc. ACM SIGCOMM*, 2013, pp. 423–434.
[17] D. Zats, T. Das, P. Mohan, D. Borthakur, and R. Katz, "DeTail: Reducing the flow completion time tail in datacenter networks," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 42, no. 4, pp. 139–150, 2012.
[18] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, Jan. 2008.
[19] N. Farrington, E. Rubow, and A. Vahdat, "Data center switch architecture in the age of merchant silicon," in *Proc. 17th IEEE Symp. High Perform. Interconnects (HOTI)*, Aug. 2009, pp. 93–102.
[20] G. Wang *et al.*, "c-Through: Part-time optics in data centers," in *Proc. ACM SIGCOMM*, vol. 40. 2010, pp. 327–338.
[21] A. Greenberg *et al.*, "The cost of a cloud: Research problems in data center networks," in *Proc. ACM SIGCOMM*, 2008.
[22] C. Guo *et al.*, "BCube: A high performance, server-centric network architecture for modular data centers," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 39, no. 4, pp. 63–74, Oct. 2009.
[23] H. Rodrigues, J. R. Santos, Y. Turner, P. Soares, and D. Guedes, "Gatekeeper: Supporting bandwidth guarantees for multi-tenant datacenter networks," in *Proc. USENIX WIOV*, 2011, p. 6.
[24] H. Wu, G. Lu, D. Li, C. Guo, and Y. Zhang, "MDCube: A high performance network structure for modular data center interconnection," in *Proc. CoNext*, 2009, pp. 25–36.
[25] H. Wu, Z. Feng, C. Guo, and Y. Zhang, "ICTCP: Incast congestion control for TCP in data center networks," in *Proc. 6th Int. Conf. Co-NEXT*, 2010, p. 13.
[26] B. Heller *et al.*, "ElasticTree: Saving energy in data center networks," in *Proc. NSDI*, vol. 3. 2010, pp. 19–21.
[27] H. Ballani, P. Costa, T. Karagiannis, and A. Rowstron, "Towards predictable datacenter networks," in *Proc. ACM SIGCOMM*, vol. 11. 2011, pp. 242–253.
[28] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," in *Proc. OSDI*, Berkeley, CA, USA, 2004, p. 10.
[29] J.-Y. Shin, B. Wong, and E. G. Sirer, "Small-world datacenters," in *Proc. 2nd ACM Symp. Cloud Comput.*, 2011, p. 2.
[30] K. Chen *et al.*, "OSA: An optical switching architecture for data center networks with unprecedented flexibility," in *Proc. NSDI*, 2012, p. 18.
[31] L. Popa, A. Krishnamurthy, S. Ratnasamy, and I. Stoica, "*FairCloud:* Sharing the network in cloud computing," in *Proc. ACM SIGCOMM*, 2012, pp. 187–198.
[32] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, "Hedera: Dynamic flow scheduling for data center networks," in *Proc. 7th USENIX Conf. Netw. Syst. Design Implement. (NSDI)*, 2010, p. 19.
[33] M. Alizadeh *et al.*, "Data center TCP (DCTCP)," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 40, no. 4, pp. 63–74, 2010.
[34] M. Shreedhar and G. Varghese, "Efficient fair queueing using deficit round robin," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 25, no. 4, pp. 231–242, Oct. 1995.
[35] M. Alizadeh, A. Kabbani, T. Edsall, B. Prabhakar, A. Vahdat, and M. Yasuda, "Less is more: Trading a little bandwidth for ultra-low latency in the data center," in *Proc. 9th USENIX Conf. Netw. Syst. Design Implement. (NSDI)*, 2012, p. 19.
[36] M. Alizadeh *et al.*, "pFabric: Minimal near-optimal datacenter transport," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 43, no. 4, pp. 435–446, Oct. 2013.
[37] N. Farrington *et al.*, "Helios: A hybrid electrical/optical switch architecture for modular data centers," in *Proc. ACM SIGCOMM*, 2010, pp. 339–350.
[38] N. Dukkipati and N. McKeown, "Why flow-completion time is the right metric for congestion control," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 36, no. 1, pp. 59–62, 2006.
[39] N. R. Adiga *et al.*, "Blue Gene/L torus interconnection network," *IBM J. Res. Develop.*, vol. 49, pp. 265–276, Mar. 2005.
[40] R. N. Mysore *et al.*, "PortLand: A scalable fault-tolerant layer 2 data center network fabric," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 39, no. 4, pp. 39–50, 2009.
[41] R. Alverson, D. Roweth, and L. Kaplan, "The Gemini system interconnect," in *Proc. HOTI*, Aug. 2010, pp. 83–87.
[42] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The Google file system," *ACM SIGOPS Oper. Syst. Rev.*, vol. 37, no. 5, pp. 29–43, 2003.
[43] T. Lam *et al.*, "NetShare: Virtualizing data center networks across services," Dept. Comput. Sci. Eng., Univ. California, Berkeley, CA, USA, Tech. Rep., 2010.
[44] V. Vasudevan *et al.*, "Safe and effective fine-grained TCP retransmissions for datacenter communication," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 39, no. 4, pp. 303–314, 2009.
[45] T. Wang *et al.*, "NovaCube: A low latency torus-based network architecture for data centers," in *Proc. IEEE GlobeCom*, 2014.
[46] T. Wang, Z. Su, Y. Xia, Y. Liu, J. Muppala, and M. Hamdi, "SprintNet: A high performance server-centric network architecture for data centers," in *Proc. IEEE ICC*, Jun. 2014, pp. 4005–4010.
[47] T. White, *Hadoop: The Definitive Guide*. Sebastopol, CA, USA: O'Reilly, 2012.
[48] Y. Chen, R. Griffith, J. Liu, R. H. Katz, and A. D. Joseph, "Understanding TCP incast throughput collapse in datacenter networks," in *Proc. 1st ACM Workshop Res. Enterprise Netw.*, 2009, pp. 73–82.



TING WANG (S'–) received the B.Sc. degree from the University of Science and Technology Beijing, Beijing, China, in 2008, and the M.Eng. degree from the Warsaw University of Technology, Warsaw, Poland, in 2011. In 2012, he interned as a Research Assistant with the Institute of Computing Technology, Chinese Academy of Sciences, Beijing. He is currently pursuing the Ph.D. degree with the Hong Kong University of Science and Technology, Hong Kong. His research interests include data center networks, cloud computing, green computing, and software defined network.



ZHIYANG SU (S'–) received the B.E. degree in computer science and technology from the China University of Geosciences, Beijing, China, in 2009, and the M.S. degree in computer network and application from Peking University, Beijing, in 2012. He is currently pursuing the Ph.D. degree with the Hong Kong University of Science and Technology, Hong Kong. His research interests focus on software defined networking (SDN) and data center networking, in particular, on improving the performance of SDN.

YU XIA (M'–) is currently a Post-Doctoral Fellow with the Department of Computer Science and Engineering, Hong Kong University of Science and Technology, Hong Kong. He received the Ph.D. degree in computer science from Southwest Jiaotong University, Chengdu, China. He was a joint Ph.D. Student and Visiting Scholar with the Polytechnic Institute of New York University, Brooklyn, NY, USA. His research interests include high performance packet switches, data center networks, and network architectures.

MOUNIR HAMDI (F'–) received the B.S. degree from the University of Louisiana, Lafayette, LA, USA, in 1985, and the M.S. and Ph.D. degrees from the University of Pittsburgh, Pittsburgh, PA, USA, in 1987 and 1991, respectively, all in electrical engineering. He is currently the Chair Professor with the Hong Kong University of Science and Technology, Hong Kong, and was the Head of the Department of Computer Science and Engineering. He is the Dean of the College of Science, Engineering and Technology with Hamad Bin Khalifa University, Doha, Qatar. He was on the Editorial Board of various prestigious journals and magazines, including the IEEE TRANSACTIONS ON COMMUNICATIONS, the IEEE COMMUNICATION MAGAZINE, *Computer Networks*, *Wireless Communications and Mobile Computing*, and *Parallel Computing*. He was a Guest Editor of the IEEE COMMUNICATIONS MAGAZINE, a Guest Editor-in-Chief of two special issues of the IEEE JOURNAL ON SELECTED AREAS OF COMMUNICATIONS, and a Guest Editor of *Optical Networks Magazine*. He has chaired over 20 international conferences and workshops, including the IEEE International High Performance Switching and Routing Conference, the IEEE GLOBECOM/ICC Optical Networking Workshop, the IEEE ICC High Speed Access Workshop, and the IEEE IPPS HiNets Workshop, and has been on the Program Committees over 200 international conferences and workshops. He was the Chair of the IEEE Communications Society Technical Committee on Transmissions, Access and Optical Systems, the Vice Chair of the Optical Networking Technical Committee, and a member of the ComSoc Technical Activities Council. He was a recipient of the Best Paper Award at the IEEE International Conference on Communications in 2009 and the IEEE International Conference on Information and Networking in 1998. He also supervised the Best Ph.D. Paper Award among all universities in Hong Kong.