Schemes for Fast Transmission of Flows in Data Center Networks

Roberto Rojas-Cessa, Yagiz Kaymak, and Ziqian Dong

***Abstract*—In this paper, we survey different existing schemes for the transmission of flows in Data Center Networks (DCNs). The transport of flows in DCNs must cope with the bandwidth demands of the traffic that a large number of data center applications generates and achieve high utilization of the data center infrastructure to make the data center financially viable. Traffic in DCNs roughly comprises short flows, which are generated by the Partition/Aggregate model adopted by several applications and have sizes of a few kilobytes, and long flows, which are data for the operation and maintenance of the data center and have sizes on the order of megabytes. Short flows must be transmitted (or completed) as soon as possible or within a deadline, and long flows must be serviced with a minimum acceptable throughput. The coexistence of short and long flows may jeopardize achieving both performance objectives simultaneously. This challenge has motivated growing research on schemes for managing the transmission of flows in DCNs. We describe several recent schemes aimed at reducing the flow completion time in DCNs. We also present a summary of existing solutions for the incast traffic phenomenon. We provide a comparison and classification of the surveyed schemes, describe their advantages and disadvantages, and show the different trends for scheme design. For completeness, we describe some DCN architectures, discuss the traffic patterns of a DCN, and discuss why some existing versions of transport protocols may not be usable in DCNs. At the end, we discuss some of the identified research challenges.**

***Index Terms*—data center network, flow scheduling, flow completion time, flow deadline, TCP incast, data center congestion-control schemes.**

I. Introduction

The computing services provided by data centers are continuing to gain users' acceptance because these services provide unparalleled mobility and ubiquitous data accessibility [1]. Data centers are large warehouses that host a very large number of servers. These servers may be used to provide different services to businesses and users in general and to benefit from economy of scale to reduce computational costs. The allocation of large clusters of servers enables elastic scalability for computing, communications, and other business applications. Software as a Service (SaaS) and Infrastructure as a Service (IaaS) are examples of high-demand services [2].

The data center network (DCN) is a critical component of the infrastructure of a data center because it enables intra-data center communications among the large amount of computing resources [3]. The properties of the DCN and the mechanisms for managing the traffic passing through it largely define the operation and performance of the data center [4]. Moreover, the economic viability of a data center greatly depends on running the data center at acceptable (if not high) performance levels and allowing users and applications to highly utilize and share its infrastructure and resources [5].

The performance of a data center is directly associated with how fast computing jobs, which are requested by users, are completed [6]. The services provided by a data center involve processing the requested computing jobs and exchanging the data associated with these jobs among servers [7]. The traffic generated by data center applications must be transmitted within a limited time to provide a timely response to the user. The traffic of DCNs is deconstructed into flows. Herein, a data center flow is defined as a sequence of packets that are generated by an application and sent from a source node to a destination node [8]. These data center flows must be transmitted (i.e., completed) in a timely manner [9].

The server clusters, the architecture of the DCN, and data center applications in combination generate traffic that follows the Partition/Aggregate model [10], [11]. In this model, a task is partitioned and processed by several servers for increased performance and efficiency. More specifically, the computing task is distributed on a tree of servers, as Figure 1 shows, where a server acts as the Top-Level Aggregator (TLA) or the root, servers at the intermediate levels of the tree act as Mid-Level Aggregators (MLAs), and servers at the leaf-level, or workers, perform the actual computational tasks [11]. The TLA receives a user request and partitions the workload among MLAs. In a similar fashion, MLAs distribute their tasks across workers. Each worker generates a response and sends it to the upper level of the tree for aggregation. The MLAs and TLA perform response aggregation, in that order. Then, the aggregated response is forwarded to the user. In the scenarios where a request for a computing task (e.g., processing data or access to data) requires a response within a deadline, each of the components of the process tree inherits portions of the deadline. Figure 1 shows an example of the deadlines associated to each component of the tree, as indicated in parentheses. Online Data Intensive (OLDI) [11], [12] applications are examples of typical applications that have time constraints (e.g., 300 ms latency). These time requirements may be stated in service-level agreements, and they are incorporated into the schemes for managing the transmission of the flows in a DCN [13]–[15].

Short and long flows are the two major groups of traffic in DCNs [6], [16], [17]. Tasks that go through the Parti-

R. Rojas-Cessa and Y. Kaymak are with the Networking Research Laboratory, Department of Electrical and Computer Engineering, New Jersey Institute of Technology, Newark, NJ 07102. Email: {rojas, yk79}@njit.edu.

Z. Dong is with the Department of Electrical and Computer Engineering, New York Institute of Technology, New York, NY. Email: ziqian.dong@nyit.edu.

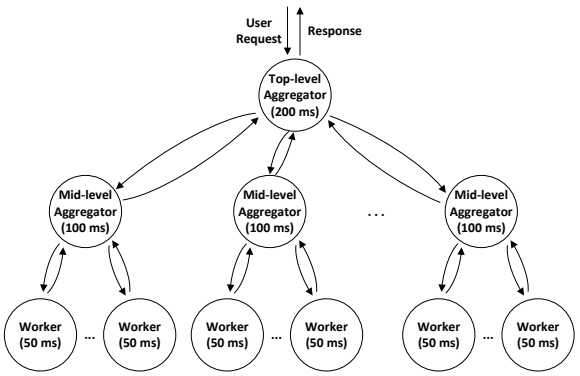

Fig. 1. Example of Partition/Aggregate traffic model [11].

tion/Aggregate model generate short flows. Short flows have a size of a few kilobytes and are associated with users' tasks, including server requests and responses. Long flows have a size of several megabytes and can be considered to be information stored in a data center or used for maintenance. Short and long flows differ in the size and the nature of their data but also in their service requirements. Short flows are time sensitive, and long flows are throughput sensitive [17]. In instances where short flows are associated with deadlines for producing a response, a data center must comply with those (or with the largest number of) deadlines. In the case where short flows are not associated with deadlines, the transmission of flows must be finished as soon as possible [11]. The time required to transmit a flow is referred to as the Flow Completion Time (FCT) [9]. Conversely, long flows must be transmitted at a satisfactory throughput to maintain the data center and keep the information up to date. The requirements of these two groups of flows must be satisfied without conflicting with each other.

The use of the Transmission Control Protocol (TCP) as a transport protocol for data center traffic may be challenged by the required small FCT of short flows because 1) the algorithms used in TCP are aimed at avoiding both congestion in the network and overwhelming the receiver; they are not engineered to minimize the FCT of a flow 2) TCP indiscriminately applies the same set of algorithms to short and long flows [18].

In addition, TCP may suffer from throughput collapse caused by the incast phenomenon that emerges in DCNs [16], [19], [20]. Incast occurs when flows from different senders converge in the same switch simultaneously. A large number of data center applications may generate incast traffic [21]–[23]. The large number of packets arriving as incast traffic in the switch may overflow the switch's buffer and cause severe packet dropping. These losses generate long waiting times for acknowledgments and trigger the retransmission of TCP segments, which together make the throughput collapse. This phenomenon greatly increases the FCT of short flows. The occurrence of this phenomenon has also prompted interest in finding strategies to prevent or alleviate it.

Because DCNs are rather administratively autonomous, a proprietary set of protocols may be used for intra-data center communications while leaving the TCP/IP stack for communications between the data center and external users [24]. The flexibility in protocol selection in a data center and the demand for satisfying the different requirements of short and long flows have motivated the search for schemes to manage data center traffic.

Providing both fast responses and high throughput is the general objective of the newly proposed schemes [25]. However, the large volume of traffic, the distribution of traffic in the data center, and the co-existence of short and long flows make achieving these objectives difficult [26].

Different DCN topologies have been proposed to improve the scalability and communications bandwidth [3], [6], [27]–[32]. However, the challenges of keeps achieving small FCTs, satisfactory flow throughput, and tolerance to incast traffic prevail because of Partition/Aggregate traffic, and these requirements seem to be rather oblivious of the network topology. One may think of recurring to the variants of TCP proposed in recent years [33]–[38] for satisfying the requirements of short and long flows, but the traffic in DCNs is different from what motivated the design of these schemes, which may not be effective.

Several schemes have been proposed to minimize the FCT of flows in data centers [9]–[11], [39]–[56]. Some of the schemes are referred to as protocols by many authors, but for simplicity, we refer to them herein as schemes. In this paper, we survey schemes that are designed to transport data center flows. For completeness, we present popular DCN architectures. We also discuss the patterns of data center traffic and the applications that generate them. On the surveyed transport schemes, we describe their operation and their properties. We base our discussion on an overall classification of schemes according to their performance objectives, such as schemes aiming to reduce FCT, also called deadline agnostic schemes, and those aiming to finish the transmission of flows within their required deadlines, also called deadline aware schemes, as Figure 2 shows. Furthermore, we classify the surveyed schemes based on their working principles, their performance objectives, and other parameters, such as complexity and compatibility with TCP, and we examine their algorithmic and hardware novelties where they apply; we list their strengths and weaknesses. We also cover the schemes designed to overcome the TCP incast problem observed in data centers. The comprehensive overview of these schemes provides information to the readers so they can select schemes that can be adopted to satisfy different performance goals in a data center and to indicate areas that may need further research. Among other information, we indicate whether the surveyed schemes have been designed in an academic or industrial environment. At the end, we comment on the areas that have been identified for research and as design goals to develop new schemes.

The remainder of this paper is organized as follows. In Section II, we present some of the existing data center architectures. In Section III, we discuss data center traffic and some of their properties. In Section IV, we describe the objectives and operation of existing schemes for reducing the flow completion time in DCNs. In Section V, we present a brief overview of the TCP incast problem and summarize the

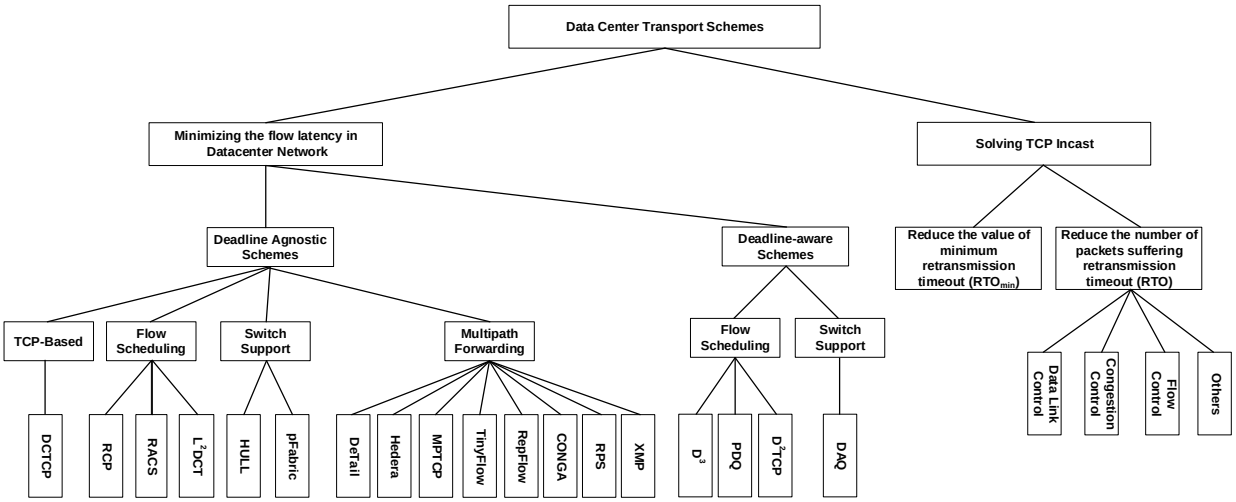

Fig. 2. Classification of surveyed schemes.

techniques and schemes used to mitigate it. In Section VI, we present a comparison of the principles and properties of the discussed schemes. In Section VII, we present some of the TCP versions and discuss why these TCP versions may not be suitable for transmission of data center traffic. In Section VIII, we present our conclusions.

II. Data Center Network Architectures

DCN architectures can be classified as switch-centric, server-centric, or hybrid structures [57]–[61]. A switch-centric architecture uses switches to perform packet forwarding, whereas the server-centric architecture uses servers with multiple Network Interface Cards (NICs) to act as switches in addition to performing other computational functions. Hybrid architectures combine switches and servers for packet forwarding [58]. In this section, we introduce the three-tier (or two-tier) [62], fat-tree [3], and VL2 [6] networks as examples of switch-centric DCN architectures, CamCube [63], [64] as an example of a server-centric DCN architecture, and BCube [27] and DCell [28] as examples of hybrid DCN architectures.

A. Three-Tier

The three-tier DCN architecture is considered a straightforward approach to building a DCN [62]. This architecture typically consists of three layers: access, aggregation, and core layers, as Figure 3 shows. In this network, servers are connected to the DCN through edge-level switches, and servers are placed in racks in groups of 20 to 40. Each edge-level switch is connected to two aggregation-level switches for redundancy. These aggregation-level switches are further connected to core-level switches. Core switches serve as gateways and provide services such as firewall, load balancing, and Secure Socket Layer offloading [58], [62]. The major advantage of this DCN is the simplicity of the topology at the expense of complex equipment and cabling. The major drawbacks of this architecture are the high cost, the low energy efficiency of the networking equipment, and the lack of agility and scalability [61].

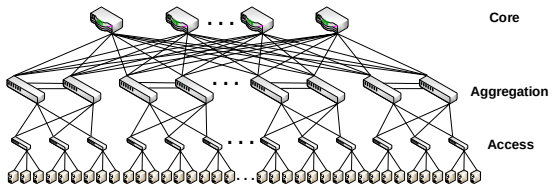

Fig. 3. Three-tier DCN.

B. Fat-Tree

The fat-tree network is a highly scalable and cost-effective DCN architecture that aims to maximize end-to-end bisection bandwidth [3], [61]. A bisection is created by partitioning a network into two equally sized sets of nodes. The bandwidth of a bisection is found by summing all of the link capacities between two partitions and the smallest bandwidth of all those partitions is the bisection bandwidth [65]. The fat-tree network is a switch-centric architecture that can be built using commodity Gigabit Ethernet switches with the same number of ports to reduce the hardware cost. The size of the network is a function of the number of switch ports k. The network is formed by edge, aggregation, and core layers. In the edge layer, there are k pods, or groups of servers, each with $k^2/4$ servers. Figure 4 shows an example of a fat-tree network with 4 pods.

Each edge switch is directly connected to $k/2$ servers in its pod. The remaining $k/2$ ports of an edge switch are connected to $k/2$ aggregation switches. The total number of core switches in the DCN is $(k/2)^2$, and each of the core switches has one port connected to each of the k pods. A fat-tree network with k-port commodity switches can accommodate $k^3/4$ servers in total. One advantage of the fat-tree topology is that all switches are identical and possibly economical. This advantage may represent economic savings in equipment cost and a simplified architecture. Another advantage is the high fault-tolerance provided by the use of multiple alternative paths between end

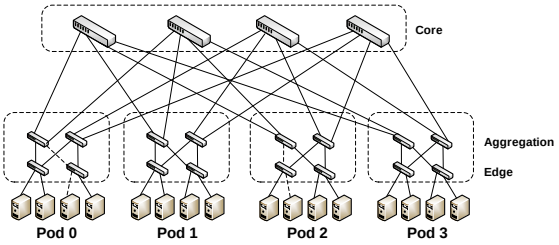

Fig. 4. Fat-tree network with 4 pods and 16 servers.

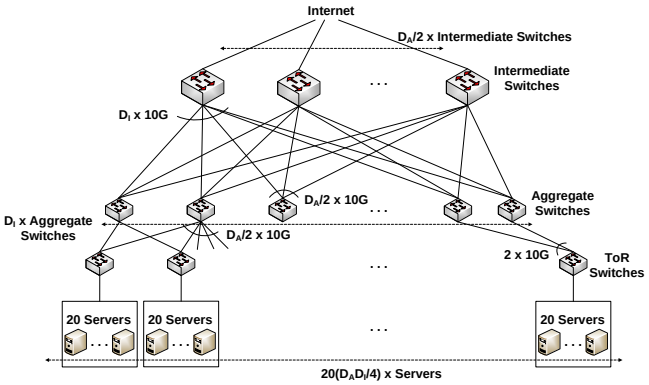

Fig. 5. VL2 network.

nodes. A disadvantage of the fat-tree architecture is the use of large numbers of switches and the increased cabling costs [61].

C. VL2

The VL2 network is a hierarchical fat-tree based DCN architecture [6], [61]. Figure 5 shows a simple VL2 network. This switch-centric network uses three different types of switches: intermediate, aggregation and ToR switches. The VL2 network targets the use of commodity switches. This network uses $(D_A)/2$ intermediate switches, D_I aggregation switches and $(D_A)(D_I)/4$ ToR switches. Intermediate and aggregation switches have different number of ports; D_I and D_A, respectively. The number of servers in a VL2 network is $20(D_A)(D_I)/4$. VL2 also employs a load-balancing technique called Valiant Load Balancing (VLB) to uniformly distribute the traffic among the network paths. One of the advantages of this architecture is its cost effectiveness due to the use of commodity switches throughout the network. Another advantage of VL2 is the ability to exploit the high bisection bandwidth because of the employed VLB technique.

D. CamCube

The CamCube network is a server-centric architecture, proposed for building container-sized data centers [63]. CamCube uses a 3D-Torus topology to directly interconnect the servers [60]. Figure 6 shows a 3D-Torus with 64 servers. CamCube, as a torus-based architecture, exploits network locality by placing the servers close to each other to increase communication

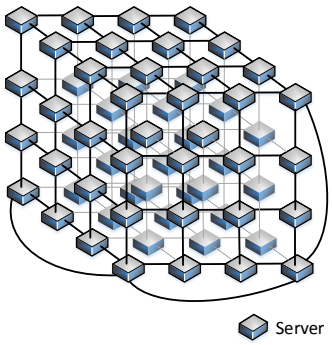

Fig. 6. 3D-Torus with 64 servers.

efficiency. CamCube may reduce costs on network equipment (i.e., switches and/or routers) by using only servers to build the DCN. This approach may also reduce costs for cooling the network equipment. CamCube allows applications used in data centers to implement routing protocols through the CamCube Application Program Interface (API). The use of this application may result in achieving higher application-level performance. On the other hand, CamCube requires multiple NICs in each server to assemble a 3D Torus network. Furthermore, the use of the Torus topology by CamCube may result in long paths, or $O(N^{1/3})$, where N is the number of servers [60]. Therefore, it has been claimed that routing complexity may be high [60].

E. DCell

The DCell network is a hybrid architecture; it uses switches and servers for packet forwarding, and it may be recursively scaled up to millions of servers [28], [61]. DCell uses a basic building block called $DCell_0$ to construct larger DCells (i.e., $DCell_1$, $DCell_2$, etc.). In general, $DCell_k$ ($k > 0$) is used to denote a level-k DCell that is constructed by combining $n+1$ $DCell_{k-1}$s, where n denotes the number of servers in $DCell_0$. $DCell_0$ has n ($n \leq 8$) servers and a commodity switch to interconnect them. Moreover, each server in a $DCell_0$ is directly connected to a server in a different $DCell_0$. The interconnection of all $DCell_0$s forms a complete graph (i.e., every pair of $DCell_0$ in the network is interconnected) if each $DCell_0$ is considered as a large virtual node. Figure 7 shows a $DCell_1$, constructed with five $DCell_0$s and 4-port commodity switches.

The main advantage of DCell architecture is its high scalability, enabled by the recursive structure. DCell is also cost efficient because it uses commodity switches and servers to perform packet forwarding. The two main disadvantages of DCell are the long communication paths between two servers in the network and the additional NICs required for each server and the associated increased cabling costs.

F. BCube

BCube is another hybrid DCN architecture that can scale up through recursion [27], [61]. BCube employs servers and commodity switches as forwarding elements and is proposed for

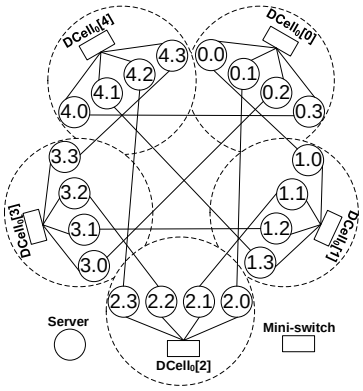

Fig. 7. $DCell_1$ constructed with 5 $DCell_0$s ($n = 4$) [28].

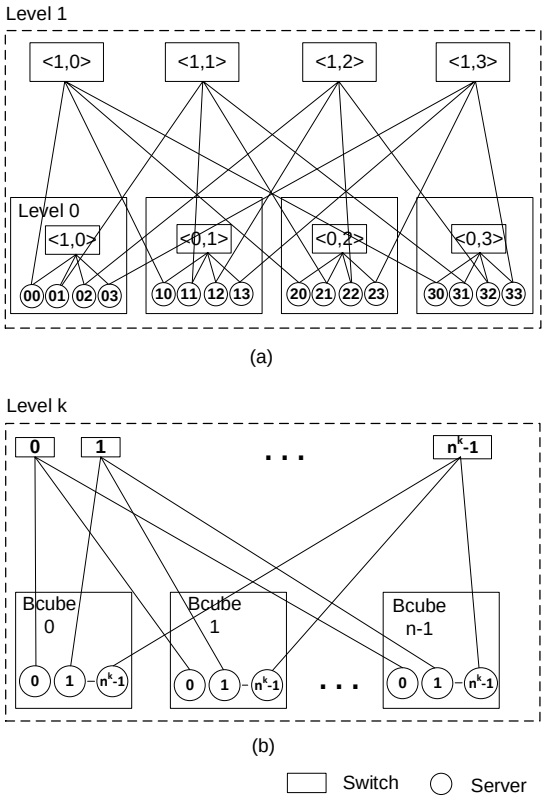

Fig. 8. **(a)** $BCube_1$ with $n = 4$. **(b)** A $BCube_k$ network.

building so-called container data centers. The scalability of this structure is limited (up to thousands of servers) compared with fat-tree, VL2, and DCell. Conversely, BCube provides high bisection bandwidth and a graceful degradation of throughput under equipment failures [27]. As a recursive approach, BCube uses $BCube_0$ as a building block, which simply consists of n servers connected to an n-port switch. In BCube, n $BCube_0$s and n n-port switches build a $BCube_1$ network. In general, a $BCube_k$ ($k > 0$) is constructed by combining n $BCube_{k-1}$s and n^k n-port switches. In a $BCube_k$, there are n^{k+1} $k+1$-port servers and $k+1$ layers of switches. Figure 8 (a) shows $BCube_1$ with $n = 4$, and Figure 8 (b) shows a $BCube_k$ network. BCube is cost effective, provides high bisection bandwidth, and yields fault-tolerance under equipment failures. However, BCube has limited scalability, and its cabling cost is high because of the numerous interconnections among switches and servers. Furthermore, the number of NICs in a server in BCube is proportional to the depth of the network [61].

There are other DCN architectures, in addition to the ones presented in this survey; these are: MDCube [66], HyperBCube [67], JellyFish [68], PortLand [69], SprintNet [70], FiConn [71], FlatNet [72], OSA [73], c-Through [74], Helios [75] and Small-World [76] for the interested reader.

These DCN architectures show a common property: clusters of servers are interconnected through an aggregation switch. This property shows that DCN applications are supported to delegate jobs to a multitude of servers. This part of the DCN, the aggregation switches, is where the many-to-one traffic is commonly observed [77], [78]. It is also clear that the Partition/Aggregate traffic may show great similarity in all of these DCN architectures and that the transport protocols may share similar challenges, independent of the adopted DCN architecture.

III. Data Center Traffic

Data centers host a wide variety of applications and services. Each application or service exhibits a different type of traffic pattern, such as many-to-one, one-to-many, and one-to-one, among the servers in the data center [16], [17], [78]–[80]. Web page creation, content composition for social networking, and web searches generate many-to-one, i.e., Partition/Aggregate traffic [10], [81]. In these applications, a Hypertext Transfer Protocol (HTTP) request is broken down into several requests by a front-end web server to fetch the partial responses for a web page from many data center workers. Upon receiving the replies from the workers, the front-end web server merges them and sends the final web page back to the user [10], [79], [81]. In that case, workers generate many-to-one traffic toward the front-end web server. For instance, a particular HTTP request for a Facebook page requires 88 cache lookups, 35 database lookups, and 392 backend remote-procedure calls from multiple servers across the data center [81]. Moreover, a simple web search request may access more than 1000 servers to form the final page [82].

The MapReduce programming paradigm aims to parallelize a large number of computations, thus enabling to process and generate large data sets [21]. MapReduce is used for many different purposes such as building search indices, capacity planning, optimizing product behavior, and large-scale machine learning [21], [81], [83]. MapReduce generates many-to-one traffic, in which intermediate key-value pairs are transferred to reducers (i.e., servers assigned in the reduce phase of a MapReduce job by the master node) and all of the values that are associated with an individual key are assigned to a single server during the shuffle stage [21], [84]–[88]. Although not generated by a single application or service, a large amount of MapReduce traffic may appear to be an all-to-all traffic pattern [81]. Therefore, many-to-many traffic

may be decomposed into many-to-one or one-to-many traffic. Furthermore, distributed file systems, such as Google File System (GFS) and Hadoop [88], fetch multiple chunks from multiple servers to form a file. This process generates many-to-one traffic [21], [87], [88]. In addition, a chunk is replicated onto multiple chunk servers (the default number of servers to keep the replicated chunk is three) for fault tolerance, which generates one-to-many traffic [21].

Email servers of Microsoft Windows Live and Hotmail generate one-to-one traffic when they receive a request for authentication from a user [89]. This traffic pattern occurs when a mail server sends an authentication request for the user to a separate authentication service residing in another server [89]. Virtual Machine (VM) migration may also generate one-to-one traffic pattern between two servers in a data center with virtualized environments [90]. For instance, VMWare's VMotion migration system allows moving a VM from one physical machine to another, thereby generating one-to-one traffic [90], [91].

Cloud computing data centers facilitate a vast variety of applications and services [2]. These different applications or services enabled by cloud computing used for the management of virtualized data centers may generate many different traffic patterns [92], [93]. However, the discretionary traffic patterns generated by cloud computing are out of the scope of this paper.

In summary, one-to-many and many-to-one are the most frequently observed traffic patterns in data centers. Of these patterns, the many-to-one traffic seems to present the most difficult challenge as flows may compete for and exhaust resources at the converging DCN switch [16], [17], [77], [78].

There are two primarily prevalent groups of applications and services in DCNs [10], [53], [55]: The first group includes bulk-data transfers, such as storage synchronization, and MapReduce-like jobs, which generate large flows [11], [46], [53]. The second group is short-lived communications (i.e., short flows) that are generated by applications or services, such as web content composition or web search [10], [46]. Moreover, an empirical study of network traffic in ten data centers revealed that short flows (e.g., flows smaller than 10KB in size) constitute 80% of data center flows and the top 10% of the long flows by size carry most of the bytes in a data center [78]. This study also showed that traffic among servers in a data center follows an ON-OFF behavior with heavy tailed distributions.

It is worth noting that all of the surveyed schemes in this paper address the traffic that stays inside the data center (i.e., intra-data center flows).

IV. Minimizing Flow Completion Time of DCN Flows

In this section, we review several of the recently proposed schemes for the transport of data center flows, such as Data Center TCP (DCTCP) [10], [94], Rate Control Protocol (RCP) [9], [39], Router Assisted Capacity Sharing (RACS) [40], Low Latency Data Center Transport (L^2DCT) [41], High-bandwidth Ultra-Low Latency (HULL) [42], pFabric [43], [44], DeTail [45], Hedera [46], Multipath TCP (MPTCP) [47], [48], TinyFlow [49], RepFlow [50], Congestion-Aware Load Balancing (CONGA) [51], Random Packet Spraying (RPS) [52], Explicit Multipath Forwarding (XMP) [53], Deadline-Driven Delivery (D^3) [11], Preemptive Distributed Quick (PDQ) Flow Scheduling [54], Deadline-Aware Datacenter TCP (D^2TCP) [55], and Deadline Aware Queue (DAQ) [56]. We present them in two different categories: deadline agnostic and deadline aware schemes.

A. Deadline Agnostic Schemes

1) Data Center TCP (DCTCP): DCTCP [10], [94] is a scheme that changes the interpretation of Explicit Congestion Notification (ECN) [95] messages at the sender to control TCP's congestion window for data center traffic. DCTCP uses the algorithms used in TCP, except those for congestion control. It differs from TCP in the way in which the sender interprets the ECN messages. DCTCP issues ECN messages in proportion to the extent of congestion in the network rather than using ECN messages to simply indicate congestion [95].

ECN is typically used with an Active Queue Management (AQM) technique such as Random Early Detection (RED) at switches/routers. ECN uses a field in the IP header with two bits, called ECN codepoints, to inform the receiver that end hosts are ECN-capable and about the incipient congestion. The ECN codepoint "11" is assigned to indicate congestion and is called the Congestion Experienced (CE) codepoint. Any router along the path between the source and the destination sets the CE codepoint if its average queue length is above a predefined threshold. In this case, the receiver generates an ACKnowledgement (ACK) packet marked with an ECN-Echo flag (ECE) in the TCP header to reflect the encountered congestion upon receiving the packet with the CE codepoint set. The sender's TCP reacts by halving the congestion window (cnwd) and reducing the value of the slow-start threshold (ssthresh) [95].

However, DCTCP reacts to congestion differently than TCP does. A sender in DCTCP modifies the congestion window according to the extent of the congestion. DCTCP aims to keep a small buffer occupancy at the switches along the source-to-destination paths. At the same time, DCTCP provisions buffer space for incast traffic bursts to reduce the number of dropped packets. By keeping low-buffer occupancy at switches, DCTCP may provide low latency, high throughput, and high burst tolerance using shallow-buffered switches. Figure 3 shows the operation of DCTCP in three different parts of the DCN:

- At switches: At the switch queues, there is a threshold K to indicate the desired level of occupancy. The CE codepoint in the IP header is set if the queue occupancy is greater than K. The value of K can be modified during a congestion episode to alleviate congestion.
- At receivers: A DCTCP receiver keeps a TCP state variable, called the DCTCP Congestion Experienced (DCTCP.CE) flag, to determine whether to mark an ACK packet with the ECE flag to indicate the encountered congestion at the switch. The DCTCP.CE flag is initially

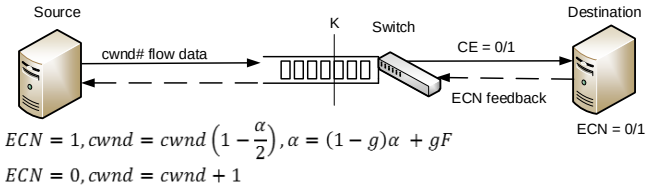

Fig. 9. The DCTCP scheme.

assigned to zero, and the receiver processes the CE codepoint in any received packet as follows:
1. If the DCTCP.CE flag at the receiver is one and the CE codepoint in the received packet is not set, the receiver sends an ACK packet with the ECE flag set for every unacknowledged packet and resets the DCTCP.CE flag.

2. If the DCTCP.CE flag at the receiver is zero and the CE codepoint in the received packet is set, the receiver sends an ACK without marking the ECE bit for every unacknowledged packet and switches the DCTCP.CE flag to one.

3. Otherwise, the CE codepoint in the received packet is simply ignored and the receiver sends one ACK packet for every m received packets to support delayed ACKs. The ECE flag of these ACK packets is set if the DCTCP.CE flag is one.

- At senders: The sender estimates the fraction of sent bytes, α, using ECE-marked ACK packets per Round-Trip Time (RTT) using:

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

where g, $0 < g < 1$, is a constant weight and F is the fraction of packets marked in the last window of data transmission (which is approximately one RTT). A large α indicates a high level of congestion. Once α is updated, the congestion window is adjusted: $cwnd = cwnd\ (1 - \alpha/2)$.

2) Rate Control Protocol (RCP): RCP [9], [39] is a congestion-control scheme that may decrease FCT. Unlike the congestion-control mechanisms used in TCP, RCP seeks to maximize the throughput or utilization at the bottleneck link. RCP is introduced to alleviate congestion in general networks and also finds its application for fairly sharing the bandwidth among data center flows. This scheme aims to approximate the behavior of a Processor Sharing (PS) [96] scheduler, which shares bandwidth of bottleneck links fairly among contending flows.

It is well known that the Shortest Remaining Processing Time (SRPT) scheduling policy achieves optimal scheduling in terms of Average FCT (AFCT) on a single link by selecting the flows with the shortest remaining processing time [97]–[99]. In RCP, the emulated PS scheduling achieves a performance close to SRPT in terms of FCT, with no flow size information required beforehand [9]. In this way, RCP achieves high link utilization while keeping a near-zero queue occupancy at the switch without a priori knowledge about flow sizes. However, it should be noted that PS achieves suboptimal AFCT compared with SRPT [40].

The working mechanism of RCP is described as follows. RCP assigns a rate, $R(t)$, to every concurrent flow defined as

$$R(t) = R(t-d_0) + \frac{[\alpha(C - y(t)) - \beta\frac{q(t)}{d_0}]}{\hat{N}(t)} \tag{2}$$

where d_0 is the moving average of the measured RTT of all packets, $R(t-d_0)$ is the previously updated rate, C is the link capacity, $y(t)$ is the input traffic rate during the last update interval, $q(t)$ is the instantaneous queue size, and $\hat{N}(t)$ is the estimated number of ongoing flows, calculated as $\hat{N}(t) = \frac{C}{R(t-d_0)}$. Here, α and β are stability and performance parameters that affect the convergence of $R(t)$, where $0 < \alpha,\ \beta < 1$. In (2), $C - y(t)$ is the available bandwidth and $\beta\frac{q(t)}{d_0}$ is the amount of bandwidth needed to empty the queue.

Figure 10 shows an example of the operation of RCP. This scheme follows four steps: 1) Every switch estimates $R(t)$, which is updated once per control interval (e.g., average RTT). 2) At the source, every packet header carries a field of the desired rate R_p. Initially, $R_p = \infty$. Once the desired rate is received by a switch, $R_p = R(t)$, if $R(t) < R_p$, and it remains unchanged otherwise. 3) The destination copies the granted rate, R_p, which equals the minimum of the assigned rates by the switches along the path, into the ACK packet and sends it to the source. 4) The source transmits at the granted rate. This rate is adjusted every estimated RTT interval.

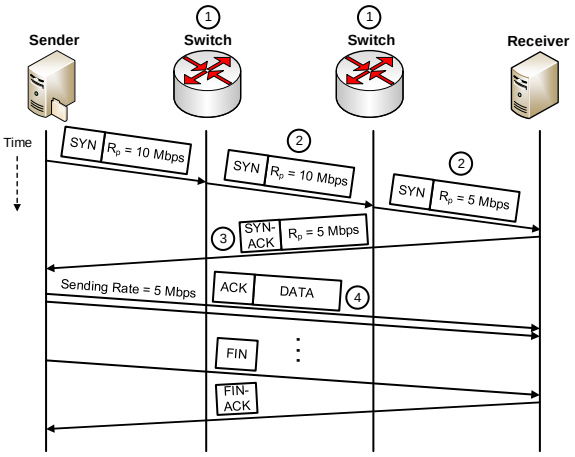

Fig. 10. Operations of RCP [39].

The implementation of RCP adopts an estimation for the number of ongoing flows, $\hat{N}(t) = \frac{C}{R(t-d_0)}$, and a user-defined parameter, τ, to determine the update rate interval, $T = min(\tau, d_0)$, which can be smaller than the average RTT. In this case, $R(t)$ is defined as

$$R(t) = R(t-T)[1 + \frac{\frac{T}{d_0}(\alpha(C - y(t)) - \beta\frac{q(t)}{d_0})}{C}]. \tag{3}$$

3) Router Assisted Capacity Sharing (RACS): Because PS scheduling is known to be sub-optimal for minimizing AFCT [39], the RACS scheme [40] was proposed to emulate the SRPT policy [97], which selects the job with the shortest remaining time first. In RACS, every flow is assigned a weight corresponding to either the remaining processing time or the residual flow size. This information is updated periodically so that routers re-allocate the rate for each flow after each update. A RACS switch allocates bandwidth in proportion to the flows' weights. The weight assigned to a flow i at time j is denoted as $w_i(j)$. The rate allocated to flow i by the router is calculated as

$$R_i = \frac{w_i(j-1)}{W(j-1)}C \tag{4}$$

where $W(j-1)$ is the weight of all flows traversing the link at interval $j-1$ and C is the link capacity of the bottleneck link. The weight of a flow can be expressed as e^{-s_r} or $1/s_r$, where s_r is the number of flow bytes remaining to be transmitted, or the remaining flow size.

Figure 11 shows the operation of RACS. The connection in the figure assumes that the flow transmission has already started. A rate request (RR) packet is sent from the source at the beginning of a control interval, k, and it carries the flow's last and current assigned weights, $w_i(k-1)$ and $w_i(k)$, respectively, and the transmission round period, t_s. When a router on the path receives an RR packet during interval k, it first updates the rate allocation for flow i, R_i, using (4), and the RTT estimation, t_r, by using a moving average. If the recently calculated R_i and t_r values are smaller than the values carried by the packet, the router updates the R_i and t_s fields in the RR packet. These updated fields are represented as R_i' and t_s' in Figure 11. The sender uses the assigned rate in the next transmission period. The destination writes the final, R_i', and t_s' into an ACK packet and sends it back to the source. The source uses these values to transmit the flow in the following transmission period and adjusts t_s based on the feedback sent by the receiver. After t_s, the source re-calculates the flow's weight and sends a new RR packet to obtain the transmission rate for the following period.

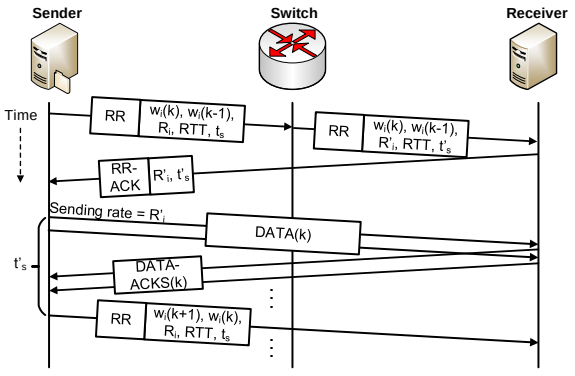

Fig. 11. Operations of RACS.

4) Low Latency Data Center Transport (L^{2}DCT): L^{2}DCT [41] schedules short flows without requiring information about flow sizes. L^{2}DCT adopts the Least Attained Service (LAS) scheduling policy [100], which emulates SRPT. However, LAS uses no knowledge of the flow size but pursues the same working principle of SRPT; to serve the smallest flow first. LAS prioritizes the flow with the least data sent and selects the flows for scheduling in a descending order of priority.

L^{2}DCT adjusts the congestion window according to the extent of congestion. The DCN switches estimate the congestion level and notify the source through ECN messages. L^{2}DCT applies a congestion avoidance algorithm similar to that in DCTCP. L^{2}DCT comprises three parts, as Figure 12 shows: a controller of the congestion window at sender, an ECN echo mechanism at the receiver, and ECN-enabled switches. At the switch side, the queue length is monitored. If the length exceeds a threshold K, the CE codepoint is set. Once destinations receive packets with the ECN-echo flag set, they issue ACKs that copies the ECN flag back to the senders.

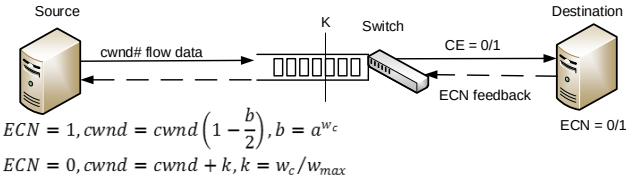

Fig. 12. The L^{2}DCT scheme.

In L^{2}DCT, a sender has two operational functions; it: 1) assigns a weight to each flow based on the number of bytes the flow has already transmitted and 2) dynamically adjusts the congestion window size according to the congestion notification (i.e., ECN). The weight assignment in L^{2}DCT follows an exponential function similar to that in RACS. However, the weight of a flow, w_c, in L^{2}DCT is determined by calculating $\frac{1}{s}$ or e^{-s}, where s represents the amount of data already served for that specific flow.

Every RTT, the sender calculates the fraction of packets marked by ECN, F, and updates α, the moving average of the fraction of ECN feedbacks, using (1). Both DCTCP and L^{2}DCT use the same exponential smoothing equation to estimate the fraction of packets marked by ECN.

As a reaction to congestion, a penalty function b is estimated:

$$b = \alpha^{w_c} \tag{5}$$

This penalty function indicates the reduction rate of the congestion window. When a sender receives an ACK packet with the ECN-echo flag set, it reduces the size of the congestion window:

$$cwnd = cwnd(1 - b/2) \tag{6}$$

To increase the congestion window size, a sender calculates the ratio of the current flow weight and the maximum flow weight, k, or:

$$k = w_c/w_{max} \tag{7}$$

where w_{max} is the maximum assignable weight, w_{min} is the minimum assignable weight, and w_c is the current flow weight, with $w_c \in [w_{min},\ w_{max}]$.

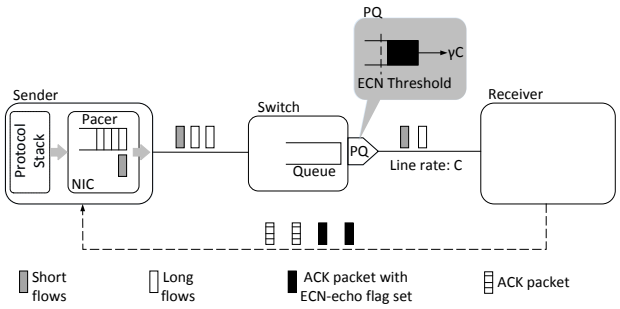

Fig. 13. Architecture of HULL [42].

A sender may increase the size of its congestion window:

$$cwnd = cwnd + k \tag{8}$$

As an example, the congestion window size is halved when congestion is severe ($\alpha = 1$, $b = 1$), as in TCP. Because short flows are assigned larger weights than those of long flows, they may use larger amounts of bandwidth than long flows.

5) High-bandwidth Ultra-Low Latency (HULL): Although several works are based on the congestion notification of DCTCP, queueing delay remains a dominant parameter. HULL [42] has been proposed to reduce queueing delays to almost zero. HULL has two goals: to achieve near baseline fabric latency and high bandwidth utilization. Most of the operations of HULL are performed at switches. These operations are: a phantom queue, DCTCP congestion control, and packet pacing. Figure 13 shows these components, which are described as follows.

a) Phantom Queue (PQ): This is a counter associated with a switch egress port. It simulates the queue buildup as it would occur in a queue of a link running at a speed lower than the actual link rate. The counter is incremented when a packet is received and decremented according to a configurable service rate (e.g., 95% of the actual link capacity). This counter aims to provide an estimate of the link utilization of the switch egress ports rather than the queue occupancy. ECN is configured at PQ using a threshold K. PQ preserves bandwidth (e.g., 5% [42]) of the link capacity to achieve zero queue occupancy at the switch buffer. This measure has the objective of reducing average and tail FCTs. When a packet exits a link, the counter's value is checked. If the counter's value is greater than or equal to K, the packet is ECN marked. Therefore, packets may be marked before the actual queue starts to build up.

b) DCTCP Congestion Control: HULL implements DCTCP at PQs. Instead of halving the congestion window when congestion is detected, as in TCP, DCTCP estimates the extent of the congestion from the fraction of ACKs received by the sender in one RTT. DCTCP has been shown to be an effective way to reduce AFCT while ensuring a high link utilization (e.g., 94% [94]).

c) Packet Pacing: HULL uses a hardware-based pacer module at the NIC of the sender nodes. The pacer sets the transmission rate of the output link [101]. The packet pacer is used to reduce burstiness and the high rates of long flows. The pacer module consists of a token-bucket rate limiter with an adjustable sending rate and a flow-association table that stores information about flows associated with the pacer. HULL updates the pacing rate (i.e., at token-bucket rate), R_{tb}, every T_r time, using the following controller:

$$\mathrm{R}_{tb} = (1-\eta)\mathrm{R}_{tb} + \eta\frac{M_r}{T_r} + \beta Q_{tb} \tag{9}$$

where η and β are positive constants (e.g., the values suggested are $\eta = 0.125$ and $\beta = 16$ [42]), Q_{tb} is the current backlog of the token bucket, in bytes, M_r is the number of bytes that the pacer receives from a host, and T_r, although not defined in the original description [42], may be considered as the sampling period for counting M_r. If an ACK packet with the ECN-echo bit set is received, Hull associates the corresponding flow with the pacer with probability p_a (i.e., 1/8 [42]). The association is dissolved after a time period, T_i. This probabilistic policy makes short flows unlikely to be paced.

6) pFabric: pFabric [43], [44] is proposed with the goal of achieving both high fabric utilization and near-optimal FCT for latency-sensitive short flows using a simple approach. pFabric aims to minimize AFCT by approximating the SRPT policy [97], [98]. It decouples flow scheduling from rate control. This is, switching nodes with small buffers (with a size smaller than twice the bandwidth-delay product) determine which packets are scheduled or dropped. These decisions are based on the assigned flow priorities. pFabric has three main components: flow prioritization, pFabric switch, and rate control. They are described as follows.

a) Flow Prioritization: For each flow generated at the source, a number encoding the priority of the flow is added into the header of each packet of the flow. The flow priority is set to the remaining flow size to be transmitted.

b) pFabric Switch: A pFabric switch executes two mechanisms: priority scheduling and priority dropping. Priority scheduling dequeues the buffered packet with the highest priority first. Priority dropping drops arriving packets if the buffer is full and the queued packets hold equal to or higher priority than that of the arriving packet. Otherwise, it drops one or more of the lowest priority packets in the buffer and buffers the arriving packet. To achieve these two functions, the switch maintains two data structures: a) a queue for actual packets, maintained in RAM, and b) a metadata queue for storing a hash of 5-tuple values to represent the flow ID and the priority numbers of the packets. When an arriving packet is stored in the buffer, the packet is added to the end of the queue and the metadata queue is updated. A binary tree of comparators is used to find the packet with the lowest priority that is to be dropped.

For the dequeuing operation, there are two steps. First, a binary tree of comparators is used to find the packet with the highest priority. Once the packet is found, the flow ID of the packet is used for a second search to find the earliest arrived packet of the selected flow. The second search is achieved by applying a bitwise parallel comparator on all the packets in the meta-data queue. pFabric avoids starvation of packets by dequeuing the earliest packet of the flow with the highest priority. This starvation would occur if the selection of a packet considered only flow priority.

c) Rate Control: In the case of a high load and multiple long flows contending for a downstream link, packet loss may occur. This scenario would eventually lead to throughput collapse in the fabric. To overcome this problem, pFabric adopts a simple rate control mechanism that inherits the congestion control mechanisms of TCP but with some variations, such as changing the initial congestion-window size, disabling fast retransmit, using Selective ACKs (SACKs), and using a fixed number of consecutive timeouts to detect congestion in the network. Specifically, flows in pFabric start with a large initial congestion window size (equal to the bandwidth-delay product of the link). The increase of the congestion window follows an additive increase for each ACK, as TCP does. Once a timeout is experienced, a multiplicative decrease is performed. pFabric also disables duplicate ACKs or any other mechanism that may trigger the fast retransmit algorithm. Moreover, it uses SACKs and keeps track of consecutive timeouts to indicate congestion. If five consecutive timeouts are experienced, the flow enters a probe mode in which sender periodically sends probe messages and returns to the slow-start phase after receiving an ACK packet.

7) DeTail: DeTail [45] is a cross-layer scheme aimed at reducing long tail FCT of short flows in a DCN. It performs cooperation between multiple layers (from data-link to application layers) and exchanges cross-layer information. Every layer implements a different function. A lossless fabric [102] is created through the adoption of a flow control at the data-link layer. A lossless fabric is one in which packet drops are avoided through the use of a flow control mechanism and storage. At the network layer, DeTail performs per-packet adaptive load balancing among alternative shortest paths that are less congested. Because no packets are dropped in cases of congestion in DeTail, an ECN-like mechanism is applied at the transport layer. This mechanism sends congestion notifications, based on the occupancies of the switch buffer, to reduce the sending rate of low-priority deadline-insensitive TCP flows. Moreover, the application layer provides flow priorities based on the latency sensitivity of the flows to prioritize delay-sensitive flows over background (long) flows. The operations of DeTail at the different layers, as Figure 14 shows, are described as follows.

a) Data-Link Layer: Priority Flow Control (PFC) [103] is used at DeTail's data-link layer to realize zero packet dropping. PFC is adopted by recent Ethernet switches [104]. Because DeTail performs occupancy estimation on both ingress and egress switch ports, it adopts a Combined Input-Output Queued (CIOQ) [104], [105] switch architecture, which has queues at the ingress and egress ports. At every port of a DeTail switch, strict priority queueing is enabled to ensure that high-priority packets are served before low-priority packets. At each ingress and egress queue, there is a byte counter to monitor the queue occupancy. If the counter exceeds a threshold, a pause message is sent to the previous hop to temporarily stop the transmission of flows with specific priorities. Once the count falls below the threshold, a resume message is sent to continue the transmissions of the indicated and previously paused priorities.

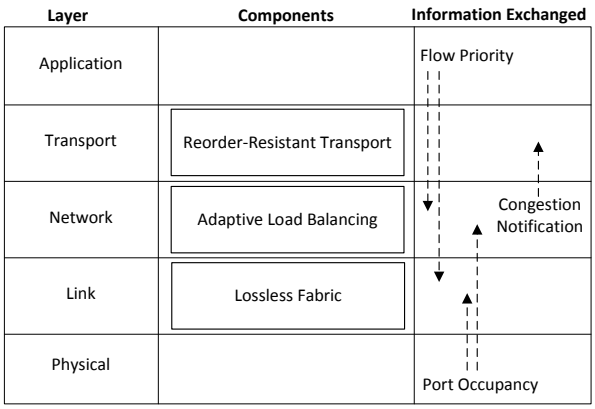

Fig. 14. Cross-layer operation of DeTail [45].

b) Network Layer: DeTail's network layer balances the load based on the congestion state of eligible paths. In other words, after arriving in a switch, a packet is forwarded to the shortest and least congested path. This process is divided into two steps: 1) The forwarding engine identifies the possible shortest and least congested paths between source and destination. 2) Packet priority and egress port occupancy are used to select the least congested paths and the switch selects an output port to forward the packet.

c) Transport Layer: DeTail uses a retrofit of TCP NewReno [106] as transport protocol. Because the data-link layer implements lossless flow control, congestion control is only triggered by ECN marks on low priority queues. DeTail disables the fast recovery and fast retransmit algorithms of TCP NewReno to avoid reacting to packets delivered out of sequence.

d) Application Layer: This layer provides the priority of latency-sensitive flows. Although eight different flow priorities are available, DeTail only uses two different priority levels based on how sensitive to latency the flows are.

8) Hedera: Hedera [46] is a flow scheduling scheme to reduce the occurrence of traffic hot spots in DCNs. It reduces large FCTs caused by network congestion and utilizes the path diversity of DCN topologies, which are multi-rooted trees with many equal-cost paths between servers. Different from other schemes, Hedera considers long flows to be the major cause of network congestion.

The operation of Hedera follows three steps: 1) It detects long flows at the edge switches of the DCN. Hedera adopts a fat-tree network as DCN. If a flow occupies at least 10% of the link capacity, it is called a long flow. 2) A central controller estimates the bandwidth demand of long flows and uses two placement algorithms to compute suitable paths for them, the Global First Fit (GFF) and Simulated Annealing (SA) algorithms. The demand estimator is a max-min fairness [107] algorithm, which runs iteratively until the flow capacities converge. In each iteration, flow capacities of source nodes are increased and the exceeded capacities at the receivers are decreased. 3) Paths are installed on the switches. Hedera uses OpenFlow-enabled switches. OpenFlow [108] uses the central controller to access and modify the forwarding tables

of switches.

For flow placement, GFF linearly searches all possible paths containing the links that may accommodate the flow being relocated and it greedily selects the first suitable path that has enough bandwidth to satisfy the flow rate. In GFF, the central controller installs the selected path by creating forwarding entries in the corresponding edge and aggregation switches.

SA probabilistically searches suitable paths for the flow being relocated to find a near-optimal solution. SA is a well-known probabilistic search method to find the global minimum of a cost function in which there may be several local minima [109]. In order to reach that goal, SA brings the system to a state with a minimum possible energy. SA starts the computation of the path by selecting a core switch. A one-to-one mapping between core switches and destination hosts in a pod defines the initial state s. The energy of the current state, $E(s)$, denotes the total exceeded capacity over all the links in the current state s, which is minimized during the runtime of SA. Here, temperature T represents the remaining number of iterations the algorithm performs and it is decremented in each iteration until it reaches zero. It is also used to calculate the *acceptance probability*, which is the probability of having a transition from the current state, s, to a neighboring state, s_N. When T is high and the energy of the neighboring state is larger than the energy of the current state, the acceptance probability may be large [109].

During the path search, Hedera selects a switch neighbor of a core switch as a member of a new path(s), s_N, and then proceeds to estimate the total exceeded capacity, $E(s_N)$, using path s_N. The path to the destination is updated by selecting the neighbor core switch, if the new path implies a smaller energy. Specifically, if $E(s_N) < E(s)$, $e_B = E(s_N)$ where e_B is the path with the smallest amount of exceeded link capacity. Hedera reduces the search space of SA by assigning a single core switch for each destination, rather than assigning one for each flow.

9) Multipath TCP (MPTCP): MPTCP [47], [48] is a data center transport scheme that utilizes the available bandwidth of a DCN by load balancing the traffic among multiple paths. This scheme is considered an extension of TCP. MPTCP is based on the assumption that one or both end hosts are multi-homed and multi-addressed. MPTCP splits a flow into several sub-flows and transmits these sub-flows on different paths. These sub-flows are transmitted between a source-destination pair, where they use either the same pair of IP addresses with different ports or different IP addresses hosted at the two ends. The transmission of each sub-flow uses a TCP connection.

Figure 15 shows an example of the basic operation of MPTCP. The figure shows the connection establishment, sub-flow setup, and connection tear down. During the connection setup, SYN, SYN/ACK, and ACK packets carry an MP_CAPABLE field to the receiver host. This field serves two purposes: 1) It is used to verify whether the other end host supports MPTCP. 2) It allows the hosts to exchange a 64-bit key (i.e., one key for the sender and another for the receiver) for authentication and to establish additional sub-flows [48]. If a source adds a new sub-flow between any pair of available addresses, it first notifies the destination about the new sub-flow by using SYN, SYN/ACK, and ACK packets with the MP_JOIN option. Figure 15 shows a sender starting a subflow from sthe ender's Port 2 to the receiver's Port 1 using MP_JOIN to indicate the new sub-flow setup. The identification is verified by using the keys exchanged in the previous MP_CAPABLE handshake. Once the subflow connection is established, data packets are transmitted as in TCP. When the transmission of a sub flow is completed, a TCP-like closing mechanism is used. However, the closing of the transmission of one sub-flow may not mean that the transmission of the whole flow has finished. When there is no more data to send, the MPTCP connection tear down process is triggered, using DATA_FIN and the corresponding DATA_ACK packets.

10) TinyFlow: TinyFlow [49] aims to regulate the traffic rather than to perform congestion control in DCNs. This approach addresses the head-of-line blocking problem of short flows and low bandwidth utilization that Equal-Cost Multi-Path (ECMP) forwarding faces under multipath routing in some DCN topologies, such as fat-tree [3] and VL2 [6]. Because ECMP does not differentiate short from long flows, it may assign the same path to both of them. As a result, short flows may experience long queueing delays at the egress buffers of switches. In addition to the head-of-line blocking problem, a low utilization of the available bandwidth may be experienced due to static mapping of flows to paths. These issues may be caused by hash collisions at switches. Assignment of two or more long flows to the same output port may congest the port.

TinyFlow mitigates these problems by breaking down long flows into short flows and randomly distributing them among all possible paths using ECMP. For multipath routing, TinyFlow uses long-flow detection and dynamic random re-routing. OpenFlow switches perform long-flow detection on the edges of the DCN. Switches perform sampling every 100 Kbyte and a long flow is detected if two samples of the same flow are found within 500 μs. Dynamic random re-routing randomly changes the egress port for the detected long flow after every 10 Kbyte of data. According to reported simulations, 18% and 40% speedup may be achieved for the mean and 99th percentile FCT of short flows, respectively, and 40% in both mean and tail FCT for long flows over ECMP [49]. Moreover, applications and end hosts do not need to be modified. Additionally, TinyFlow may be coupled with a congestion control scheme to improve performance of the DCN.

11) RepFlow: The design objective of RepFlow [50] is to reduce the FCT of short flows without requiring any change in TCP at end hosts and the switches in the DCN.

The application of ECMP depends on the selection of one outgoing link among the equidistant paths to the destination in a switch. Although ECMP is an efficient technique, it does not differentiate between short and long flows. Therefore, ECMP may increase the FCT of short flows.

RepFlow replicates short flows to decrease the probability of head-of-line blocking. In this scheme, each sender establishes a TCP connection to the receiver in addition to the original one if the flow size is equal to or smaller than 100 Kbytes and

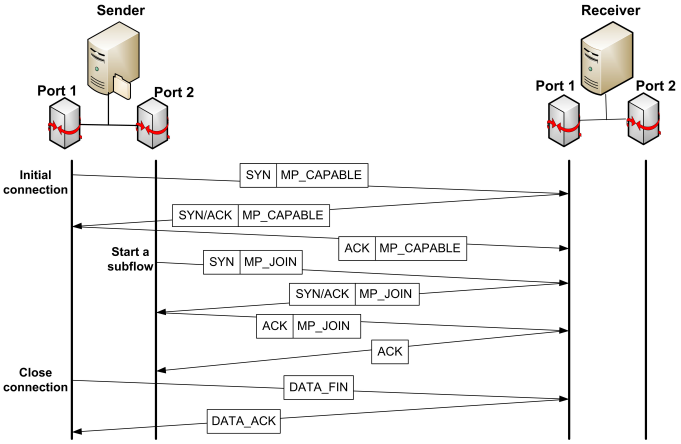

Fig. 15. Example of the operation of MPTCP.

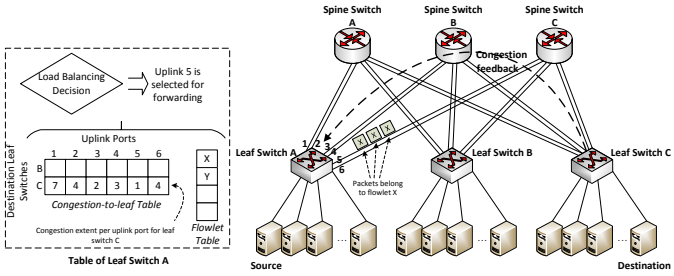

Fig. 16. CONGA architecture.

sends identical packets using both connections. In this way, the original flow is replicated. The multiple connections are differentiated by the use of different port numbers. Therefore, the hashing performed by ECMP likely produces two different paths for two identical flows but with a different 5-tuple (i.e., source and destination IP addresses, port numbers, and protocol). By replicating flows, RepFlow exploits the multipath nature of the DCN topology to decrease FCT for short flows.

It has been reported that the overhead generated by the replication of short flows is negligible because short flows only account for a small fraction of the traffic in the system. Because RepFlow is not a congestion-control-based scheme, it may be combined with another data center transport scheme, such as DCTCP. Moreover, RepFlow does not require any modification at end hosts or the switches in DCN.

12) Congestion-Aware Load Balancing (CONGA): CONGA is a distributed, congestion-aware, and in-network load-balancing scheme for DCNs. This scheme aims to balance the traffic without modifying TCP. The design of this scheme holds the following objectives: 1) to provide a fast reaction (e.g., tens of microseconds) to congestion; 2) to operate obliviously to transport protocols (e.g., TCP or UDP) and without requiring modifications to them; 3) to make the scheme robust to link asymmetries and link failures; 4) to make it configurable in switches, such that the scheme may be applied throughout or in a portion of the DCN; and 5) to be optimized for 2-tier DCN topologies. CONGA may decrease the FCTs of all flows and it may provide higher throughput compared to some other multipath forwarding schemes, such as ECMP and MPTCP. CONGA splits flows into flowlets [110], estimates the congestion on the paths based on the feedbacks from destination leaf (i.e., edge) switches, and assigns the flowlets to the least congested paths. A flowlet is defined as a bursts of packets of a flow that are separated by sizable gaps [51], [110]. The use of flowlets decreases the possibility of out-of-sequence packet delivery and provides a finer load-balancing granularity as compared to flow-based load balancing [51].

The functionality of CONGA depends on switches in a 2-tier leaf-spine topology (a 2-tier leaf-spine topology is a 2-level DCN topology consisting of edge and core switches with 2:1 oversubscription at the leaf level, as Figure 16 shows). In CONGA, each leaf (i.e., edge) switch maintains a congestion-to-leaf table, to store the congestion feedbacks received from destination leaf switches. Destination leaf switches collect congestion information carried in data packets from the source switches and send these congestion feedbacks back to sender leaf switches on the traffic flowing in the opposite direction. To forward a flowlet, a leaf switch checks its congestion-to-leaf table and selects the uplink with the minimum congestion. The architecture of CONGA is depicted in Figure 16.

Congestion measurement in CONGA is performed by an estimator function, called Discount Rate Estimator (DRE), which is run on all switches in the DCN. DRE measures the load of a link by incrementing a register value X for each packet sent through the link by the size of the packet and decrementing it every $T_{dre}\mu s$ using a multiplicative factor, α, following $X = X(1-\alpha)$. Note that X here, is proportional to the traffic rate, R, over the link. Specifically, $X \approx R\tau$, where $\tau = T_{dre}/\alpha$. The selections of τ, and T_{dre} are performed experimentally [51]. The congestion metric is calculated as $X/\tau C$, where C represents the link capacity.

Leaf switches also perform flowlet detection and bookkeeping. A load balancing decision takes place only for the first packet of a flowlet. The subsequent packets of the flowlet follow the same path. Therefore, the table is used to keep track of the flowlets while they remain active. At the same time, flow detection is achieved by a timer, called flowlet inactivity timer, which periodically (every T_{fl} seconds) times out and deletes the entry of a flowlet from the flowlet table, if there is no packet to send for that flowlet in a period equal to $2T_{fl}$.

13) Random Packet Spraying (RPS): Random Packet Spraying (RPS) [52] is a forwarding technique that forwards packets of flows through different shortest paths to their destinations, where a path is randomly selected with uniform probability. Unlike ECMP, RPS performs packet-by-packet forwarding at switches of a multi-rooted tree topology, such as a fat-tree [52]. Instead of hashing the five-tuple header fields of each flow to find out the path for forwarding the flow as performed by ECMP, RPS randomly selects an egress port for each packet from the alternative equal-cost paths to the destination and forwards the packet. Note that out-of-sequence packet delivery may occur if RPS is employed as the forwarding scheme at switches because of the potentially different latencies of the alternative paths. In the case of out-of-sequence packets, TCP's congestion control triggers

the congestion avoidance phase and halves the congestion window. This TCP response leads to suboptimal performance because TCP does not distinguish out-of-sequence packets from lost packets [111]. However, RPS is designed under the assumption that alternative equal-cost paths have similar queueing properties and latencies because of the symmetry of these paths [52]. Latencies of multiple equal-cost paths are assumed to be similar when RPS is employed throughout the DCN because RPS may distribute the load equally among all the equal-cost paths. Although out-of-sequence packet delivery may still occur in RPS, the performance of RPS is higher than that of ECMP in terms of throughput and flow completion time.

RPS also uses a modified version of the RED scheme called Selective-RED (SRED) to keep the queue lengths of a switch almost equal under the scenario of a link failure. By keeping the queue lengths of a switch at equal length, RPS may maintain similar latencies on equal-cost paths and therefore, the out-of-sequence packet delivery may be avoided. Other schemes for path selection based on RPS have been considered [112].

14) Explicit Multipath (XMP) Congestion Control: Explicit multipath congestion control (XMP) [53] is a congestion-control scheme for MPTCP. It aims to balance two seemingly conflicting goals: providing high throughput for long flows and small latencies for short flows. XMP is based on transmitting subflows, as in MPTCP, to exploit the multipath feature of DCNs. XMP creates subflows for long flows and uses TCP for short flows. Similar to MPTCP, in XMP each subflow has its own congestion window and each of them independently experiences the congestion conditions along the path. Unlike MPTCP, XMP considers all subflows to belong to a single flow to obtain efficiency and fairness among all the flows in the DCN. XMP comprises Buffer Occupancy Suppression (BOS) and Traffic Shifting (Trash) algorithms. BOS aims to attain small latencies for short flows, using ECN. BOS marks the packets at switches along the path, as DCTCP does. Unlike DCTCP, the receiver uses two bits inside the TCP header, ECE and Congestion Window Reduced (CWR) codepoint, to inform the sender of the number of packets that have experienced congestion. This information indicates the extent of network congestion. These two bits are used to differentiate large latencies generated by the DCN from those generated by the delayed acknowledgment mechanism. A sender increases its congestion window by one maximum segment size if it receives an ACK packet with both ECE and CWR bits set to zero. Otherwise, the sender stops increasing the congestion window and starts the congestion avoidance phase. During the congestion avoidance phase, the sender reduces its congestion window by $1/\beta$ if it receives an ACK packet with both ECE and CWR bits set to one MSS. The selection of β and K values must satisfy $(K+ Bandwidth\ Delay\ Product)/\beta \leq K$ for 1 Gbps link rate and 400 μs of RTT. If both the ECE and CWR bits are zero, the sender increases the congestion window by δ after one RTT. Here, δ is an indicator of how aggressively a flow competes for bandwidth with other flows on the same path. The Trash algorithm calculates the individual transfer rate of each flow and works in combination with the BOS algorithm. The objective of the Trash algorithm is to shift the traffic of subflows on the congested paths to the less congested paths. The Trash algorithm first calls the BOS algorithm for each subflow to determine the independent sending rates. Second, a total rate for each flow is calculated by summing the assigned rates to every subflow of a flow. Third, using the total rates calculated in the second step, the individual δ parameter of each subflow is adjusted by the Trash algorithm. The Trash algorithm continues this process until rate convergence occurs for every subflow in the network.

B. Deadline Aware Schemes

1) Deadline-Driven Delivery (D^3): The D^3 [11] scheme incorporates awareness of flow deadlines for the assignment of transmission rates. As in RCP, senders calculate the requesting rate for flows before the transmission of the flow and the switches along the path to destination participate in determining the sending rate for each active flow. A sender initially calculates the requested rate, r, for each of its flows before transmitting them using $r = s/d$, where s and d indicate the flow size and the deadline of the flow, respectively.

The sender sends the requested rate to destination. Switches along the path to the destination allocate a rate to each flow and inform the sender of the granted rate by using ACK packets on the reverse path. Senders periodically (i.e., every interval t, which is approximately one RTT) ask for a new rate allocation as the traffic load may vary with time. Rate requests for the next interval are piggybacked to one of the data packets sent during the interval t.

When a switch receives these rate requests, it accepts as many as the total requested bandwidth remains smaller than or equal to the output-link capacity, C, while it greedily satisfies the requests in the order they arrive.

To maintain a high link utilization, D^3 shares the rest of the bandwidth by setting $a_{t+1} = r_{t+1} + fs$, where a_{t+1} and r_{t+1} are the allocated and the requested rate for interval $t+1$, respectively, and fs is the fair share. Here, fs is calculated as $fs = (C - D)/N$, where D denotes the total demand and N represents the total number of flows requesting transmission. Once the available bandwidth is smaller than the $t + 1$th rate request (i.e., $r_{t+1} > C - A$, where A is the total allocated bandwidth), the algorithm assigns all the remaining bandwidth to the flow by using $a_{t+1} = C - A$. D^3 applies the same approach to short and long flows.

Figure 17 shows the operation of D^3. The sender initiates a flow by sending a SYN packet carrying the rate request at interval t. The switch on the path calculates the allocated rate and appends it to the packet. The receiver sends back the allocated rate to the sender and the sender starts sending the flow at the allocated rate. During the transmission, the sender updates the rate requests at each RTT period of time to update the minimum bandwidth required for completing flows.

2) Preemptive Distributed Quick (PDQ) Flow Scheduling: D^3 grants flow rate requests in a First Come First Serve (FCFS) manner, so that it may neglect scheduling flows whose deadline may be about to expire. For instance, if a flow with a large deadline arrives earlier than one with a small

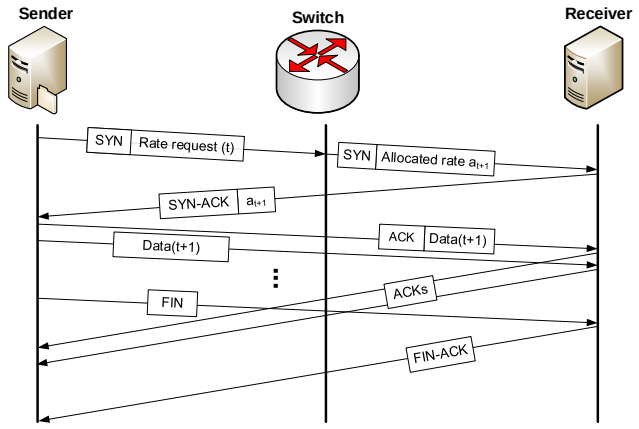

Fig. 17. Operation of D^3.

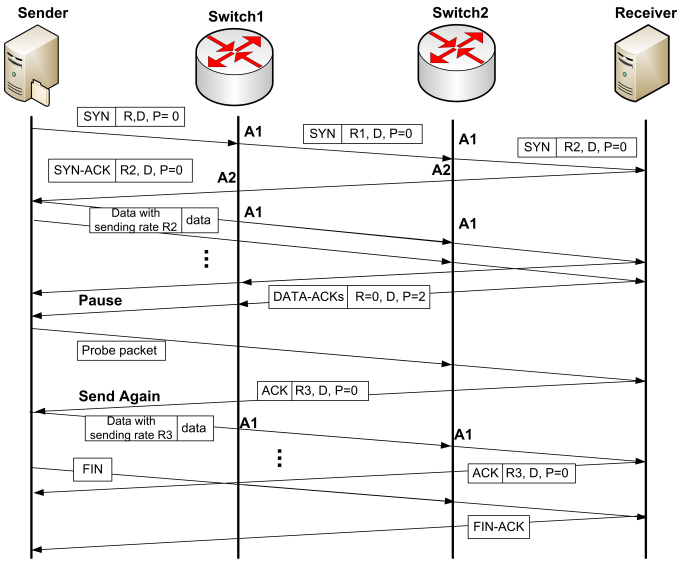

Fig. 18. Example of the operation of PDQ.

deadline, D^3 grants the flow with the larger deadline first. The selection policy of D^3 may unnecessarily delay flows with small deadlines. Therefore, the number of flows that may finish their transmission fast or on time under this policy may not be large. PDQ [54] addresses this drawback by using the Earliest Deadline First (EDF) and the Shortest Job First (SJF) selection policies to schedule flows. PDQ aims to transmit the most critical flow as soon as possible by preemptively allocating the needed resources to it. In cases where two or more flows have equivalent deadlines, PDQ assigns the highest priority to the flow with the shortest transmission time.

To realize flow-by-flow scheduling instead of packet-by-packet, PDQ records the state of every concurrent active flow at both senders and intermediate switches. PDQ switches monitor the flows' state during their transmission and when a new flow is selected among the existing flows, it takes preemptive action; it pauses the transmission of flows and assigns the link bandwidth to the new selected flow. Figure 18 shows an example of the operation of PDQ. As in D^3, packet headers in PDQ maintain a field to indicate the flow rate request when a flow connection is initiated through a synchronization (SYN) packet. However, the rate request in PDQ indicates the maximum achievable sending rate (i.e., sender's NIC line rate). Once the switch receives this request, it accepts or rejects the flow depending on the criticality of the flows contained in the flow table and the available bandwidth, $Avail$. In the example, two switches receive a SYN packet and check the deadline D of the packet to determine the flow's priority. If the priority of the packet becomes the highest among the set, a rate equal to $\min\{Avail, R\}$ is assigned to the flow. Here, R is the current flow's sending rate and is updated by the switches along the path (e.g., R1, R2, and R3 in Figure 18 are the updated sending rates, where R1 is the first accepted rate and R3 is the last accepted rate). However, if there is any switch that pauses this flow (e.g., $P = 2$ in the figure means that Switch 2 pauses it), the sender holds the transmission until a rate (larger than zero) is granted. While a flow is paused, probe packets are sent periodically until the transmission is resumed.

In addition, PDQ may add the Early Termination (ET) and Early Start (ES) schemes as additional optimizations. ET is used to terminate the flows without a feasible deadline. Due to the excessive demand from all flows, some flows are quenched in order to satisfy the service deadline of others. ET quenches a flow if one of the following conditions is true for that flow: (1) the deadline has passed, (2) the deadline is greater than the remaining transmission time as estimated by the currently assigned rate, and (3) the flow is paused at this point and the estimated RTT is longer than the time to the deadline. ES is used to improve link utilization when a switch terminates servicing one flow and starts servicing another; before scheduling a new flow, the link may be idle for one or two RTTs; the time it takes to notify the source of the new flow. ES allows the transmission of a new flow to start before the current and nearly-completed flow finishes. ES uses a threshold, K, to define nearly-completed flows; if the time needed by the current flow to finish is smaller than K RTTs, the flow is considered nearly finished.

Suppressed probing in PDQ reduces the probing frequency for the flows sending probe packets during their pause periods. PDQ also uses a rate control strategy to react to queue built up produced by ES and the loss of control packets (e.g., pause packets).

3) Deadline-Aware Datacenter TCP (D^{2}TCP): D^{2}TCP [55] aims to meet deadlines for deadline-driven applications (e.g., online data-intensive applications) and to provide high throughput for background flows. It inherits the functions of DCTCP, such as adjusting the flow congestion window size in proportion to the extent of congestion in the network. To transmit the flows with small deadlines ahead of those with large deadlines, D^{2}TCP enhances the congestion avoidance algorithm of DCTCP by adding deadline awareness to it, where priority of a flow is determined by the flow deadline. Flows are assigned an amount of bandwidth proportional to the priority of the flow.

The operation of D^{2}TCP consists of three parts, as Figure

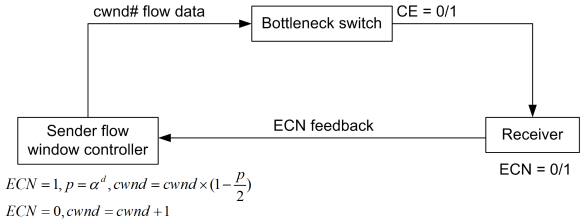

Fig. 19. Operation of D^{2}TCP.

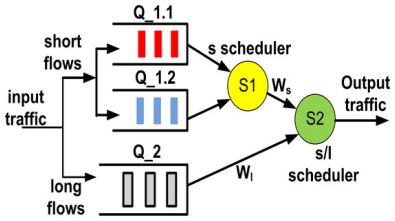

Fig. 20. Queue structure at DAQ switches [56].

19 shows: a rate controller at the sender side, ECN-enabled switches along the path, and the receivers functioning as reflection points for conveying congestion-feedback messages. In D^{2}TCP, it is assumed that the DCN uses shared-memory commodity switches. At the switch side, the ECN mechanism monitors the queue length. If the queue length exceeds a threshold K, the CE codepoint is marked. Once the receiver node receives the packets with marked CE bits, it generates ACKs for the received packets with ECN-marked feedback, and sends the ACKs back to the sender. The congestion avoidance algorithm takes place at the sender side. It decreases the flow's congestion window based on ECN feedback and the deadline imminence factor, d, which indicates the time remaining to the deadline. If no packet is ECN-marked, the sender increases the window size to probe for available bandwidth. As in DCTCP, a weighted average that measures the extent of congestion is maintained as in (1). After the calculation of α, a penalty value, p, is calculated as:

$$p = \alpha^d \tag{10}$$

Here, p indicates that the congestion window must be reduced as:

$$W = \begin{cases} W(1-\frac{p}{2}) & \text{if } p > 0 \\ W+1 & \text{if } p = 0 \end{cases} \tag{11}$$

The estimation of d follows $d = \frac{Tc}{D}$, where T_c is the time needed to complete the flow's transmission in a deadline-agnostic manner, and D is the remaining time until the deadline of the flow expires. T_c is calculated by an approximation, $T_c = B/\frac{3W}{4}$, where B and W are the remaining size of the flow and the current congestion window size of the sender, respectively.

4) Deadline Aware Queue (DAQ): DAQ aims to quickly serve short flows and guarantee bandwidth (or throughput) for long flows [56]. This scheme is deadline aware, so it expects the issuing application to associate a deadline to each short flow. To achieve FCTs within deadlines, DAQ uses switches along the path to provide differentiated service to flows that need to be transmitted *urgently*. There are two levels of urgency, determined at the packet level (not exclusively at the flow level) for short flows: urgent and not urgent. Long flows are not analyzed for urgency because they are not latency sensitive. Packets of short flows with a deadline smaller than a certain threshold are denominated urgent packets and those packets with a deadline equal to or larger than the threshold value are not considered urgent. The supporting switches use three queues to differentiate service: a) an urgent queue for urgent packets, b) a non-urgent queue, and c) a queue for long-flow packets. The adoption of these queues (and the scheduler that selects which queue receives service at a given time) provides low complexity for this scheme, as switches are flow stateless because once a packet is sent to a queue, no information about the flow is kept. Figure 20 shows the queueing structure at DAQ switches. The queues are served by a two-level scheduler, where the first level is a weighted round-robin scheduler that gives a weight w_s to short-flow queues (urgent and not-urgent queues) and w_l to the long-flow queue. These weights guarantee a minimum service rate to long flows. The second level of the scheduling scheme used between the short-flow queues is strict priority, where the higher priority is assigned to the urgent queue.

DAQ may use ECN-like signaling to indicate congestion in the switches and optimize the performance of the scheme. In addition, DAQ uses flow control to avoid overflowing the switch queues. DAQ shows that the priority queueing structures in switches shorten the FCT of small flows and guarantee throughput to long flows.

V. Schemes for TCP Incast

TCP incast is likely to occur when a concurrent many-to-one communication pattern takes place in a DCN when multiple servers simultaneously send data to a single receiver through an intermediate switch. The instant traffic burst quickly overflows the limited switch buffer causing packet losses, which in turn triggers an overwhelming number of retransmissions after their respective time outs. The combination of time outs, packet losses, and retransmissions significantly reduces the TCP throughput. This low link utilization phenomenon is also named goodput collapse. Figure 21 shows the network segment where TCP incast in a DCN occurs. Here, many synchronized servers send data to one receiver through the same Top-of-Rack (ToR) switch. In this case, congestion occurs at the link connected to the receiver server. Many typical data center applications are based on the many-to-one traffic pattern, such as cluster storage [19], MapReduce [21], and web search applications based on Partition/Aggregate work flows [10], [12]. In this section, we describe the existing solutions to the TCP incast problem.

A. Reducing the Minimum Retransmission Timeout (RTO_{min})

Recent studies have shown that setting the minimum Retransmission TimeOut (RTO), or RTO_{min}, to a value similar to the average RTT experienced in DCNs, within the range of a few microseconds, may significantly alleviate the throughput

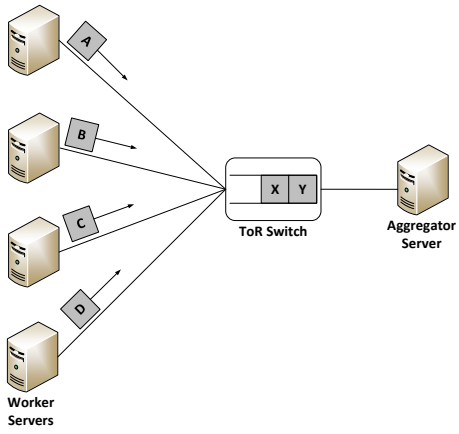

Fig. 21. Scenario of incast congestion in data center networks.

collapse caused by TCP incast [19], [113]. Other proposed approaches to solve the throughput collapse combine high-resolution timers (to set RTO values within the microsecond range), randomizing RTO values to desynchronize senders, and disabling TCP delayed ACKs [114].

B. Reducing the Number of Packets Experiencing RTO

The use of data-link-layer control techniques such as Backward Congestion Notification (BCN) [115]–[117], Quantized Congestion Notification (QCN) [118], Forward Explicit Congestion Notification (FECN) [119] and Enhanced Forward Explicit Congestion (E-FECN) [120], congestion control, and flow control mechanisms at the transport layer such as Incast Congestion Control for TCP (ICTCP) [77], DCTCP [10], ethernet flow control [121], and priority flow control [103] have been proposed to overcome the TCP incast problem.

1) Data link control: The Internet Engineering Task Force (IETF) [122] and the IEEE Data Center Bridging Task Group of IEEE 802.1 Working Group [123] are dedicating efforts to congestion notification schemes for DCNs to address the increasing bandwidth demand. Some of the outcomes are the BCN and QCN schemes, which are both queue-length-based control schemes; they sample arriving packets at the congested switch and send feedback to the sender indicating the severity of the congestion, which is determined by the experienced queue length. When a source node receives feedback notifications, it adapts the sending rate to mitigate the congestion in the network. BCN aims to maintain high throughput, minimize queueing delay, and achieve proportional fairness [117]. As another congestion-notification scheme, the goal of QCN is to provide a stable, responsive, fair and simple to implement congestion control in data-link layer [124]. Other schemes with similar objectives, such as FECN [119] and E-FECN [120] have also been proposed. These congestion-notification schemes are applied at the data-link layer.

BCN is based on the principle that congestion occurs at the core switches. These switches may be provisioned with hardware-based monitors for the detection of congestion. BCN requires collaboration of the source nodes with the congested node. The traffic source has a rate regulator (e.g., a token-bucket traffic shaper) integrated in it. Once a core switch detects congestion by monitoring the length of an output queue, it notifies the sources with a BCN message, which includes the severity level of the congestion. The sources react to the received BCN message by updating the flow transmission rates.

QCN is proposed by the IEEE 802.1Qau project. It aims to function as an Ethernet congestion-control algorithm. QCN comprises two algorithms: 1) the Congestion Point (CP), which takes place at the switches and 2) the Reaction Point (RP), which acts as a rate limiter. The objective of the CP algorithm is to maintain the buffer occupancy at a desired level. As in BCN, a QCN switch randomly samples incoming packets with probability p and calculates the level of congestion, F_b, by combining the queue size and rate excess. If the calculated F_b is smaller than the one found in the probe packet, the F_b value in the packet is updated and sent back to the sender. Once the source receives the feedback, the RP algorithm adjusts the sending rate, which is a function of F_b. The RP algorithm may increase the data rate as recovery after a rate-decrease episode. The rate increase takes place in two phases: Fast Recovery (FR) and Active Increase (AI). Every time a rate decrease occurs, RP enters the FR state. In the FR state, there are five cycles, each of which equals 150 Kbyte of data transmission by the rate limiter in RP. At the end of each cycle, the current rate, CR, is updated as $CR = (CR+TR)/2$, where TR is the sending rate before a rate decrease message is received. After five cycles, RP enters the AI state and starts probing the path for extra bandwidth. QCN also uses a timer to rapidly converge to a high sending rate when CR is very small.

QCN is more efficient than BCN, in terms of the traffic generated by feedback messages, as it only sends negative feedbacks. Despite controlling link rates in a DCN, link utilization and TCP throughput of QCN are low. These are the results of unfairness among flows under synchronized read requests from senders when TCP incast occurs [125].

FECN is a congestion control scheme for data centers to achieve extremely low or zero packet drops at congested switches. FECN outperforms BCN in both fairness and response time [120]. FECN uses proactive signaling to periodically detect the available bandwidth of the network. Specifically, the source periodically sends Rate Discovery (RD) packets that include a rate field, r, initialized to -1. Switches on the path calculate the fair share for each flow based on their load and queue length, and they advertise a rate, r_i, for the ith measurement interval. If the advertised rate for interval $i+1$ is larger than r in RD packet, it is updated to $r = r_i + 1$. Upon receiving an RD packet, the receiver reflects the packet containing r back to the sender, thus the sender sets its sending rate as advertised in the packet.

E-FECN combines the benefits of BCN and FECN. It enhances FECN by reacting to sudden traffic bursts and by allowing switches to send BCN messages to senders under severe congestion. As in FECN, E-FECN uses the RP algorithm to reduce the sending rate. FECN and E-FECN are explicit rate-based closed-loop control methods, but switches do not send FECN feedbacks directly to the sender. E-FECN allows the switches to send BCN messages (through the receivers) to

TABLE I
WINDOW ADJUSTMENT IN ICTCP

Condition	Window adjustment
$d_i^b \leq \gamma_1$	Increase the advertised window if there is enough available bandwidth on the network interface; decrease the quota after the advertised window is increased.
$d_i^b > \gamma_2$	Decrease the advertised window by 1 Maximum Segment Size (MSS) if this condition holds for three continuous RTTs.
Otherwise	Window remains unchanged.

the senders if the queue length of any switch along the path becomes greater than a threshold, indicating severe congestion.

2) Congestion control: The congestion control algorithm of TCP [111], [126] is effective to avoid packet losses in wide area networks, such as the Internet. However, its effectiveness lessens under TCP incast traffic. The ICTCP scheme [77] addresses this issue by proactively modifying the TCP advertised window (at the receiver side) before packet loss occurs. ICTCP keeps the receiver informed of the achieved throughput and the remaining available bandwidth. This scheme aims to maintain a window size small enough to keep throughput collapse from occurring and large enough to keep the transmission of data with a high throghput. The incast congestion control scheme attempts to set a proper advertised window for all TCP connections at the same bottleneck link and to share the bandwidth fairly among these flows; it adjusts the advertised window based on the ratio (b^e - b^m)/b^e, where b^m and b^e denote the measured and the expected throughput, respectively. ICTCP uses the current throughput of connection i, b_i^s, to smooth the measured throughput, $b_{i,new}^m = \max(b_i^s, \beta b_{i,old}^m + (1-\beta) b_i^s)$, where $0 < \beta < 1$ is the smoothing factor. The expected throughput, b^e, is calculated as $b^e = \max(b_i^m,\ rwnd_i/RTT_i)$, where $rwnd_i$ and RTT_i are the advertised window and the RTT of connection i, respectively.

The ratio of the throughput difference, d_i^b, is defined as $d_i^b = (b_i^e - b_i^m)/b_i^e$, such that $d_i^b \in [0,1]$. ICTCP increases the advertised window when this ratio is small, and decreases the window when the ratio is large. The thresholds γ_1 and γ_2 are used to define an increase or decrease of the window size, as Table I shows. The values of γ_1 and γ_2 are suggested as 0.1 and 0.5, respectively [77]. The available bandwidth is defined as $BW_A = \max(0,\ \alpha C - BW_T)$ where α is a parameter used to decrease the oversubscribed bandwidth during window adjustment with $\alpha \in [0,1]$, and BW_T is the total incoming traffic on that interface.

Different from adjusting the advertised window as in ICTCP, DCTCP may overcome the TCP incast problem by keeping the buffer of the switches along the workers-aggregator path with a small occupancy. DCTCP reacts to the extent of the congestion by adjusting the congestion window. The extent of congestion is indicated by the fraction of packets that ECN-enabled switches mark. This reaction is different from that of ICTCP, which adjusts the TCP advertised window to control each connection's sending rate. In ICTCP, the window size is calculated as a function of the available bandwidth of the receiver and on the difference between the measured and expected throughputs, as in TCP Vegas [127].

Although it is not explicitly stated, RACS and pFabric also alleviate the throughput collapse caused by TCP incast. RACS mitigates the problem by maintaining small queues as DCTCP does. By means of the ECN feedback mechanism and the ability to decrease the sending rate to less than one packet per RTT, pFabric exhibits higher performance that DCTCP, in terms of AFCT and packet loss rate, under a TCP incast scenario. pFabric alleviates the problem by reducing RTO_{min}, which is selected as 45 μs in experimental evaluations.

3) Flow control: Flow control mechanisms, such as those provided by the Ethernet [121] and priority flow control [103] can effectively help to manage the amount of aggregated data sent through a single switch. However, their performance in multi-switch or multi-layered switching topologies is limited because of the head-of-line-blocking problem of switches [19]. PAUSE frames used in Ethernet flow control fully stops the transmission of all flows passing through the link for a specified period of time. It should be noted that to take measures against the flow causing congestion, stopping the whole transmission on the link will also stop other flows from sharing the same link even though they may not contribute to the congestion. PFC is proposed to solve this problem, by stopping the congesting flow without affecting the other flows. However, throughput collapse may still occur in PFC. In such a scenario, the buffer of a blocked switch may overflow and subsequently block the upstream buffers, spreading the initial congestion into a saturation tree [128], [129].

4) Increasing the buffer size: A straightforward method to avoid packet loss under TCP incast is to increase the amount of switch buffers, but this measure also increases the switch cost and it may not alleviate the throughput collapse for a large number of senders. In fact, it has been shown that increasing the buffer space allocated per port on switches increases the number of senders that can transmit before incast is experienced by the system. Therefore, it may simply postpone throughput collapse [19].

VI. CLASSIFICATION, COMPARISON, AND DISCUSSION

In this section, we classify and compare the surveyed schemes according to their objectives, features, and operation.

A. Category: Scheme Objectives

Table II shows the classification of the surveyed schemes according to their objectives. These objectives are listed in the order they appear in the table.

Reducing Mean or Tail FCT: The majority of the surveyed schemes aim to decrease both the AFCT and the tail FCT for short flows, while PDQ, RACS, and L^{2}DCT aim to explicitly reduce AFCT only. DeTail is the only scheme among these that has the sole objective of reducing the Tail FCT. Although AFCT is a practical parameter to show the performance of a scheme for short flows, tail FCT (i.e., 99th or 99.9th percentile

TABLE II
CLASSIFICATION OF SCHEMES ACCORDING TO THEIR OBJECTIVES

Scheme	Reducing Mean(M)/Tail(T) FCT for Short Flows	No/Less Modification Requirement in Existing DCN Infrastructure	Fairness	Maximizing Number of Flows that Meet Their Deadlines	High Burst Tolerance	Low/Zero Buffer Occupancy	Providing High Throughput	Mitigating TCP Incast
DCTCP	√ (M and T)				√		√	√
RCP		√	√					
RACS	√ (M)				√		√	√
L^2DCT	√ (M)	√						
HULL	√ (M and T)					√	√	
pFabric	√ (M and T)	√						√
DeTail	√ (T)							
Hedera							√	
MPTCP			√				√	
TinyFlow	√ (M and T)	√					√	
RepFlow	√ (M and T)	√						
CONGA	√ (M)	√					√	
RPS							√	
XMP	√ (M)		√				√	
D^3				√	√		√	
PDQ	√ (M)			√				
D^2TCP		√		√				
DAQ				√			√	
ICTCP								√
BCN			√			√		
QCN		√	√					
FECN			√			√		
E-FECN			√					

FCT) is used to observe the worst-case performance of the system.

Small or No Modification of Existing DCN Infrastructure: Deploying the schemes on an existing DCN architecture and TCP/IP stack may require changes in the existing infrastructure, whether at the end hosts or network switches. Small and modest changes to the existing structure in the adoption of these schemes may imply faster incorporation into the data center.

Fairness: Fairness, or max-min fairness, is another measure sought by several schemes. A fair scheme may avoid resource monopolization by long flows. Such schemes may reduce the average or tail FCT of short flows [39]. Here, RCP, MPTCP, XMP, BCN, QCN, FECN, and E-FECN are listed in this category.

Maximizing Number of Flows that Meet their Deadlines: D^3, PDQ, D^2TCP, and DAQ aim to maximize the number of short flows that meet their deadlines. Schemes seeking to achieve this goal may also decrease the FCT of short flows that are not associated with deadlines [54].

High Burst Tolerance: The Partition/Aggregate traffic model that many applications in DCNs employ may generate bursts of packets in the DCN [10]. It is important for the schemes to be able to tolerate the bursts and to alleviate the congestion caused by these bursts. DCTCP, D^3, and RACS provide high burst tolerance. For instance, D^3 attains burst tolerance by assigning a base rate to the flows and allowing them to send header-only packets for a specified period of time.

Low or Zero Buffer Occupancy: Another objective is to keep the queue size at the switches small if not zero. We categorize

this objective as low or zero buffer occupancy. Having a low or zero buffer occupancy may reduce the queueing delay experienced by latency-sensitive short flows. BCN, FECN, and HULL pursue this objective. For example, HULL reserves a small portion of the actual link capacity, called bandwidth headroom, to keep the switch buffer occupancy at almost zero. This measure may help to decrease FCT of short flows. Other examples are BCN and FECN, which also aim to keep the queue occupancy at a constant level by signaling the sender node for a sending rate adjustment.

Providing High Throughput: DCTCP, RACS, HULL, MPTCP, TinyFlow, CONGA, RPS, XMP, D^3, and DAQ share the objective of maintaining high throughput for long flows in DCNs. These schemes assign lower priority to long flows than short flows. Hedera aims to maximize the aggregate network throughput (i.e., bisection bandwidth).

Mitigating TCP Incast: ICTCP, DCTCP, pFabric, and RACS report having a capability to mitigate the throughput collapse caused by TCP incast. Among these, ICTCP is the only scheme that is explicitly designed with this objective, while the other schemes aim to reduce FCT but they may also be applicable in this case.

B. Category: Used Mechanisms

Table III classifies the schemes according to their working principles. We identified the following ten mechanisms: ECN feedback, window adjustment, rate assignment (rate allocation), load balancing, preemptive scheduling, flow prioritization, flow replication, multipath forwarding, cross-layer operation, and hardware modification.

ECN Feedback: DCTCP, RACS, L^2DCT, HULL, DeTail, XMP, and D^2TCP use ECN feedbacks to decrease the sending rate of the source node as a way to reach a desirable buffer occupancy at the switches to alleviate network congestion.

Window Adjustment: ICTCP, DCTCP, PDQ, D^2TCP, and L^2DCT use this mechanism to control the rate in which flows are transmitted. In this technique, the advertised window (rwnd) or congestion window (cwnd) of the connection of a flow may be adjusted to increase or decrease the transmission rate of the flow. The window adjustment mechanism may benefit from ECN feedbacks to detect or avoid congestion.

Rate Assignment: Explicit rate assignment is a technique performed at switches to calculate and assign a transmission rate for requesting flows according to the adopted service policy. This technique enables switches along the path to explicitly select flow rates before the actual transmission starts. In this way, the aggregated rates of the flows transmitted on a path does not exceed the path capacity. Moreover, explicit rate assignment may be used to prioritize or penalize flows [11], [54]. The schemes that use this mechanism are FECN, RCP, RACS, D^3, and PDQ.

Load Balancing. Load balancing is a technique to exploit the multipath feature of DCNs that distributes traffic among possible paths. DeTail, MPTCP, TinyFlow, CONGA, RPS, and XMP use this technique.

Preemptive Scheduling: Preemptive scheduling is a technique that pauses the active flow(s) and give the service to the highest priority flow as it emerges. PDQ uses this technique.

Prioritization for Short Flows: Most schemes assign higher priority to short flows than to long flows. Prioritization results in assigning higher sending rates to flows with small deadlines associated to short flows, as observed in D^3, or by preemptively scheduling short flows first to finish them quickly, as in PDQ. Another way to prioritize short flows is to aggressively shrink the congestion window of far-deadline flows, while slightly reducing the congestion window of near-deadline flows, as proposed in D^2TCP. RACS assigns higher priorities and sending rates to short flows by considering the remaining size of a flow. Similar to RACS, L^2DCT uses the least-attained service scheduling policy to assign rates. pFabric employs a priority-based packet scheduling at switches where it uses the remaining flow size to assign a priority for each packet. DAQ dedicates queues for short flows at the switches to provide differentiated service to them.

Exploiting Multipath: Because many of today's DCNs utilize multiple paths between any pair of end hosts [3], [62], MPTCP, DeTail, RepFlow, TinyFlow, CONGA, Hedera, RPS, and XMP make use of these paths to provide a high bisection bandwidth [46], [47], decrease FCT [49]–[51], [53], increase throughput [47], [53], or perform load-balancing throughout the network [45], [47], [49], [51]–[53]. MPTCP segments a flow into subflows and forwards them through the paths selected by ECMP. DeTail uses per-packet adaptive load balancing at the network layer and selects the forwarding paths by considering the port buffer occupancies of switches. RepFlow replicates each short flow and also selects the paths by using ECMP. TinyFlow segments long flows into short flows by randomly selecting the egress port of a long flow among the equal-cost paths to the destination for every 10KB of data sent for the corresponding long flow. CONGA segments a flow into flowlets and uses uplink congestion information of leaf switches to select multiple paths. Hedera re-routes long flows that exceed 10% of the link capacity through different paths. RPS sprays the packets randomly among the equal-cost paths to destination. XMP, as in MPTCP, segments flows into subflows to use the multiple paths of DCNs.

Cross-layer Operation: Cross-layer design is a technique in which different layers of the protocol stack work collaboratively to mitigate congestion in the network or to speedup the transmission of flows. Performing layer-wise operations and cross-layer information exchange may help to mitigate congestion. DeTail adopts cross-layer schemes to reduce the long tail of FCTs. TinyFlow and DAQ use this approach to reduce the FCT of short flows, and CONGA employs it to provide a responsive congestion-aware load balancing mechanism.

Hardware Modification: Hardware modification may be used on the end hosts or DCN switches to support a desired flow transport property. HULL, DAQ, CONGA, BCN, and E-FECN resort to hardware modification. For example, HULL uses hardware-based flow pacers at end hosts and DAQ uses dedicated queues for prioritized flows at switches. CONGA requires leaf-switches to keep track of flowlets, select the best possible port for a flowlet to be forwarded by checking the congestion-to-leaf table, and generate congestion feedback. BCN and E-FECN use a hardware up-down counters in the

TABLE III
CLASSIFICATION OF SCHEMES ACCORDING TO THEIR WORKING MECHANISMS

Scheme	ECN Feedback	Window Adjustment	Explicit Rate Assignment	Load Balancing	Preemptive Scheduling	Prioritization for Short Flows	Exploiting Multipath	Cross-Layer Design	Hardware Modification
DCTCP	√	√ (cwnd)							
RCP			√						
RACS	√		√			√			
L^2DCT	√	√ (cwnd)				√			
HULL	√								√ (switch)
pFabric						√			
DeTail	√			√			√	√	
Hedera							√		
MPTCP				√			√		
TinyFlow				√			√	√	
RepFlow							√		
CONGA				√			√	√	√ (switch)
RPS				√			√		
XMP	√			√			√		
D^3			√			√			
PDQ		√ (cwnd)	√		√	√			
D^2TCP	√	√ (cwnd)				√			
DAQ						√		√	√ (switch)
ICTCP		√ (rwnd)							
BCN									√ (switch)
FECN			√						
E-FECN			√						√ (switch)

generation of feedback messages at switches.

C. Category: Scheme Features

Table IV summarizes the features of all presented schemes. We categorize the scheme features by whether they are TCP-based, fault tolerant, or able to coexist with TCP. We also indicate their layer(s) of operation and how they react to congestion.

TCP-based Operation and Operating Layers: We first list the schemes whose design is based on TCP or a variant of it. This feature indicates the layer(s) of the TCP/IP protocol stack where modifications may be needed. We note that most schemes work at the transport layer. Because many DCNs are implemented with Layer-2 switches, we find a number of schemes that operate at this layer. Moreover, we find that schemes such as DeTail, TinyFlow, CONGA, RPS, and DAQ operate across several layers.

Reaction to Congestion: The reaction of the schemes to congestion may be either reactive or proactive. Reactive schemes are mostly focused on congestion control. We find that most of the schemes are proactive, except for pFabric, Hedera, MPTCP, and D^2TCP. Proactive schemes aim to prevent congestion from occurring. Schemes that are based on rate control are also considered proactive.

TCP Coexistence: In general, schemes that can coexist with TCP in DCNs means that they may be able to interact with the protocol and mechanisms in it. Schemes that do not explicitly state whether they can coexist with TCP or do not provide sufficient information to infer whether they can coexist with TCP are labeled as ”Not known.”

Fault Tolerance: The redundancy provided by the use of multiple paths in the DCN is the major mechanism for providing fault tolerance. DeTail, Hedera, MPTCP, CONGA, RPS, and XMP are examples of schemes in this category.

TABLE IV
MAIN SCHEME FEATURES

Scheme	TCP-based	Operating Layer	Reaction to Congestion	TCP Coexistence	Fault Tolerance	Modification Requirements		
						Sender	Receiver	Switch
DCTCP [10]	√	L4	Proactive	Not known		√	√	ECN
RCP [39]		L4	Proactive	Not known		√	√	√
RACS [40]		L4	Proactive	√		√	√	√
L^2DCT [41]	√	L4	Proactive	√		√	√	ECN
HULL [42]	√	L4	Proactive	Not known		√	√	ECN
pFabric [44]	√	L4	Reactive	√		√	√	√
DeTail [45]		L2, L3, L4	Proactive	Not known	√	√	√	ECN
Hedera [46]	√	L4	Reactive	√	√			√
MPTCP [47]		L4	Reactive	√	√	√	√	√
TinyFlow [49]	N/A	L3, L4	N/A	√				√
RepFlow [50]	√	L5	Proactive	√				
CONGA [51]	√	L3, L4	Proactive	√	√			√
RPS [52]	√	L3, L4	Proactive	√	√			√
XMP [53]		L4	Proactive	√	√	√	√	√
D^3 [11]		L4	Proactive			√	√	√
PDQ [54]		L4	Proactive	Not known		√	√	√
D^2TCP [55]	√	L4	Reactive	√		√	√	ECN
DAQ [56]		L2, L4	Proactive	√				√
ICTCP [77]	√	L4	Proactive	√			√	
BCN [117]	N/A	L2	Proactive	√		√		√
QCN [118]	N/A	L2	Proactive	√		√		√
FECN [119]	N/A	L2	Proactive	√		√	√	√
E-FECN [120]	N/A	L2	Proactive	√		√	√	√

Modification Requirements: We indicate if modifications are required at the sender, receiver, or switches for the implementation of the scheme. We observed that most schemes require a level of support from DCN switches, whether with ECN or any other particular functionality. It should be noted that although ECN is given as a modification requirement in Table IV, commodity switches being currently used in DCNs are shipped with ECN capability [130].

D. Category: Operation Principles

Table V shows our classification of the schemes based on their operation principles. We list the schemes that aim to reduce FCT and also include the schemes that are aimed to increase application throughput as the number of flows that finish transmission before their deadlines over the total number of flows requesting service [11].

Some of the schemes use a combination of multiple strategies to manage the transmission of flows, and therefore, they may fit in several of the categories. We set the following major categories of operation principles: rate control, congestion control, flow control, and multipath forwarding. As the last column of the table shows, most of the schemes resort to some kind of switch operation.

Rate Control: The switches in the schemes under the rate control category generally select the sending rates of flows, and senders transmit the flows at the selected rates. The assignment of rates may follow a policy that may fulfill the objective of the scheme (see Table II). RCP emulates PS to assign a fair rate to all flows traversing the switches along the

TABLE V
OPERATION PRINCIPLES OF SCHEMES

Scheme	Features				
	Rate Control	**Congestion Control**	**Flow Control**	**Multipath Forwarding**	**Switch Operation**
DCTCP		*cwnd* adjustment			ECN
RCP	Emulating PS				Fair-share rate calculation
RACS	SRPT approximation				SRPT-based rate calculation
L^2DCT	LAS approximation				ECN + LAS-based rate calculation
HULL		*cwnd* adjustment			ECN + PQ
pFabric		TCP-based			Flow priority + Flow ID comparators
DeTail		Switch-buffer-occupancy based	Priority Flow Control	Hop-by-hop congestion-based	ECN + Drain byte counters
Hedera				Re-routing of long flows	Hash-based forwarding for new flows + OpenFlow support + Long flow detection
MPTCP				ECMP + Subflows	ECMP
Tinyflow				ECMP + Splitting long flows	
RepFlow				ECMP + Flow replication	
CONGA				Congestion-aware flowlet forwarding + ECMP at spine switches	Congestion measurement + Flowlet detection
RPS				Packet-based	Random egress port selection
XMP		*cwnd* adjustment		Subflows	ECN
D^3	Deadline-based greedy rate assignment				Calculation for rate allocation
PDQ	Queue size and measured aggregate traffic based		EDF and SJF schedulings		Flow priority comparison + Keeping active flow states
D^2TCP		*cwnd* adjustment			ECN
DAQ			L2-based		Urgent and non-urgent queue support + Waited RR
ICTCP			*rwnd* adjustment		

path. RACS and L^{2}DCT use the SRPT and LAS scheduling policies, respectively, to minimize the FCT of short flows. Switches in D^3 and PDQ calculate the sending rate of flows to increase application throughput by using greedy assignment. D^3 shares the bandwidth among the flows that arrived first and PDQ dedicates the link bandwidth to critical flows.

Congestion Control: The congestion control category includes DCTCP, HULL, XMP, and D^{2}TCP. In these schemes, the size of the congestion window is adjusted according to the ECN feedback, while D^{2}TCP considers deadline information of the flows in addition to the ECN feedback. pFabric uses TCP but it increases the initial window size and reduces RTO_{min} as a reaction to congestion. DeTail uses buffer occupancies to detect (and react to) congestion by using drain byte counters at each ingress and egress port of the switches.

Flow Control: The schemes in this category may operate on an end-to-end or hop-by-hop fashion. DeTail and DAQ use PFC and data-link-layer-based flow control techniques, respectively, to avoid buffer overflow at switches. ICTCP adjusts the receiver window to control the sending rate of the flows and PDQ employs the EDF and SJF scheduling schemes to control the flows by pausing and allowing the transmission of them at any time.

Mutipath Forwarding: Multipath forwarding exploits the multipath nature of DCNs to mitigate the effects of congestion on the network. MPTCP, RepFlow, TinyFlow, RPS, and XMP are examples of schemes that use this principle. MPTCP, RepFlow, and TinyFlow also use ECMP to find available equal-cost paths from source to destination. MPTCP and XMP create subflows to exploit multipath diversity, RepFlow replicates short flows, TinyFlow splits long flows among the calculated ECMP paths. Hedera uses a hash-based forwarding, similar to ECMP, only for new flows. Hedera's switches reroute a long flow when the flow uses 10% or more of the link capacity. DeTail uses port buffer occupancies to select a packet's next hop. CONGA selects a path to forward the flowlets based on path congestion state to perform load balancing. RPS randomly selects an egress port for each packet among the equal-cost paths to the destination and forwards them individually.

Switch Operation: It is important to note that a large number of these schemes require ECN support and most schemes require different types of switch operations, as the last column of Table V indicates. Some of the schemes such as HULL, DAQ, and CONGA require hardware modification at switches to support their operations. HULL implements a hardware pacer at the NIC of the sender to pace large flows, and a PQ at switches. DAQ maintains three different queues to service different flows (i.e., urgent short flows, non-urgent short flows, and long flows). CONGA requires each switch along the path calculates a congestion metric and each edge switch stores these calculated congestion metrics in order to select the best output port to forward each flowlet.

In the following section, we look into the properties of the schemes classified as rate control, congestion control, multipath transmission, and hardware modification categories.

E. Major Operation Principles

1) Rate Control Schemes: Schemes based on rate control pursue allocation of the rates in which flows are transmitted. The properties listed for these schemes include fairness, deadline awareness, flow prioritization, and whether a scheme responds proactively or reactively to network congestion scenarios, as Table VI shows.

Fairness: Fairness [131], [132] is sought in DCNs to avoid monopolization of resources. This objective is particularly important for cloud computing DCNs, where the resources are shared among a large number of tenants at any given time. Max-min fairness [107] is desirable for sharing bandwidth in DCNs, and therefore, fairness is considered by several data center transport schemes. However, fairness may conflict with performance objectives. For example, recent research work has shown that fairness may not be aligned with achieving high performance transport in terms of the number of flows whose (transport) deadlines are satisfied [11]. We found that several of the rate control schemes may not be fair (or explicitly hold that objective), except for RCP, which pursues fairness by emulating PS scheduling. RACS and pFabric prioritize flows according to the remaining portion of the flow to be transmitted as they aim to finish the smallest short flows first. D^3 shares the bandwidth based on the rates requested by flows in the order in which the requests arrive. PDQ, which also grants rates to flows based on their requests, allocates bandwidth by considering deadlines and flow sizes.

Deadline Aware: Some of the rate control schemes are deadline aware, and others consider that no deadlines are obtained or explicitly used (or required) by the data center applications. The performance metrics for these two groups of schemes are application throughput and AFCT. Among these schemes, D^3 and PDQ are deadline aware. There are other schemes that, though they are not in the rate control category, are deadline aware. We comment on them in the following sections.

Flow Priority: Schemes assign priority to flows to be used for selecting the flow that receives service and, in some cases, the amount of service (and therefore, of bandwidth) that the flow receives. This is also a method to select the level of sharing of resources (e.g., high priority flows are allocated more bandwidth than are low priority flows). For example, RACS and pFabric assign high priorities to the flows with small remaining sizes. Exceptions of using flow priorities are RCP and DCTCP, which consider every flow with equal priority.

Deployability: We refer to deployability as the amount of changes needed in the infrastructure of a DCN to adopt a transport scheme. In this category, RCP, RACS, and D^3 require the smallest number of changes on existing infrastructure, so their deployability is considered high. pFabric accommodates multi-path forwarding and is considered to have medium deployability. PDQ requires being able to support preemptiveness, and its deployability is considered low.

Table VII presents the scheduling policies and the storage requirement for flow states at switches. We summarize the particular function and operations needed at switches and end

TABLE VI
COMPARISON OF RATE CONTROL SCHEMES

Scheme	Fairness	Deadline Aware	Flow Priority	Deployability
RCP	Yes	No	No	High
RACS	No	No	Yes	High
pFabric	No	No	Yes	Medium
D^3	No	Yes	No	High
PDQ	No	Yes	Yes	Low

hosts, and we list the used signaling methods.

Scheduling Policy: Switches in RCP, RACS, and PDQ apply scheduling policies to select transmission rates for each flow. RACS emulates the SRPT policy by assigning flow priority and using the switches to perform differentiated service (i.e., allocation of bandwidth) according to the assigned priorities.

RCP, RACS, and D^3 update the allocated bandwidth for each flow every RTT, or time slot, based on the current queue size, aggregate incoming traffic, and the number of flows. Each time slot, bandwidth allocation is adjusted for a large number of short flows. These schemes are then designed to be stateless to avoid using a large storage for flow states, and complex and fast scheduling algorithms at supporting switches. This feature keeps their complexity low.

Switch Storage: pFabric and PDQ log every flow state, pFabric uses two (state) vectors, and PDQ uses a table of states. PDQ switches store the states of the most critical 2κ flows, where κ is the number of flows whose sending rate is greater than zero. Therefore, the space complexity of PDQ depends on the number of flows needed to fill up the link capacity, or with complexity $O(\kappa)$. When a switch receives a new flow request, PDQ compares the packet's parameters with those entries in the table to determine whether to accept or stall the flow.

Host Operations: End hosts send data at the rate specified by the switches along the sender-receiver path in many of these schemes. In addition to rate assignment by the switches, RACS hosts assign a priority to each existing flow, D^3 hosts calculate the minimum requested rate assigned to a flow, and PDQ hosts determine whether a flow is terminated early.

Signaling Method: Most of the schemes in this group use a proactive signaling method to prevent congestion, except for pFabric. pFabric uses packet dropping to trigger rate control at the source nodes.

2) Congestion Control Schemes: Schemes that aim to reduce network congestion to minimize FCT fall in this category. Schemes such as L^2DCT, D^2TCP, and HULL follow the same congestion control mechanism as in DCTCP. Tables VIII and IX show the properties of the schemes in this category. Some of the properties considered are: fairness, whether they are deadline aware or agnostic, and TCP compatibility.

Fairness and flow priority: L^2DCT, D^2TCP, and DeTail assign priorities to flows based on flow sizes or deadlines. Flows with higher priorities are assigned larger sending rates. On the other hand, DCTCP and HULL assign rates to flows similarly to TCP; they aim to achieve max-min fairness for flows over a long period of time.

Deadline aware: D^2TCP is the only deadline aware scheme in this category. The congestion control algorithms of L^2DCT and D^2TCP are very similar; these two algorithms add an exponential factor to the penalty on the congestion window size. In these two schemes, the objective is to adjust the congestion window according to the flow priority and in proportion to the extent of congestion.

Compatibility with TCP: We consider two aspects for indicating compatibility with TCP [133]: one aspect is whether the scheme is backward compatible with TCP, and the second aspect is whether existing applications need to be modified to support the new scheme. From this category, D^2TCP and DeTail are two schemes that may not be compatible with TCP. D^2TCP may require modification on the application to obtain information of flow deadlines from legacy applications. Because DeTail is a cross-layer scheme, the layer interfaces must be modified to provide the required information exchange. For example, the application layer in DeTail uses a socket interface to provide deadline information to network and data-link layers, and at the same time, the physical layer is required to notify the data-link and network layers about the occupancies of the egress ports.

3) Multipath Forwarding Schemes: The schemes in this category are MPTCP, Hedera, TinyFlow, RepFlow, CONGA, RPS, and XMP. They use multiple paths to forward flows to reduce possible congestion points in the DCN. These schemes share some common properties on how they perform control of the multiple path transmissions and load balancing. Table X shows a summary of the properties of the schemes in this category, and it indicates the target traffic of each scheme, and how the schemes are implemented.

Control Plane: Hedera is a centralized scheduler, implemented on OpenFlow [108], [134] switches to communicate with the central controller through a secure channel. The main function of the controller is to modify flow entries in the switches to redirect flows if path modifications arise. MPTCP is a TCP-compatible scheme; it is a distributed scheme (i.e., executed by the end hosts) and inherits the basic operations of TCP. TinyFlow, RepFlow, CONGA, RPS, and XMP use distributed control mechanisms where load-balancing decisions are made locally.

TABLE VII
FEATURES OF RATE CONTROL SCHEMES

Scheme	Scheduling Policy	Switch Storage	Host Operations	Signaling Method
RCP	PS	No per flow states	Sending data	Proactive
RACS	SRPT	No per flow states	Assign flow priority	Proactive
pFabric	Priority schedule, priority drop	Part of per flow states	Sending data	Reactive
D^3	Greedily accept rate requests	No per flow states	Calculate flow rate requests	Proactive
PDQ	EDF and SJF	Per flow states	Flow early termination	Proactive

TABLE VIII
COMPARISON OF CONGESTION CONTROL SCHEMES

Scheme	Fairness	Deadline Aware	Flow Priority	Burst Tolerance	Compatibility with TCP
DCTCP	Yes	No	No	High	Yes
L^2DCT	No	No	Yes	Low	Yes
D^2TCP	No	Yes	Yes	Low	No
HULL	Yes	No	No	High	Yes
DeTail	No	No	Yes	High	No

TABLE IX
DEPLOYABILITY OF CONGESTION CONTROL SCHEMES

Scheme	Switch Operations	Source Operations
DCTCP	ECN	Update α per RTT
L^2DCT	ECN	Calculate flow current weight w_c every RTT
D^2TCP	ECN	Calculate deadline imminence factor d
HULL	Phantom queues with ECN	Packet pacing for long flows
DeTail	ECN enabled at low priority queues, load balancing per packet	Sending data

Reaction to Congestion: MPTCP reacts to network congestion by linking TCP's congestion-avoidance algorithm on multiple subflows and by explicitly relocating the subflows on more congested paths to less congested ones. Hedera uses a central scheduler to monitor and replace congested (or nearly congested) paths or links. TinyFlow and RepFlow depend on ECMP to determine the lower-cost paths. They load balance their flows among those paths. CONGA reacts to congestion by using collected congestion measurements at edge switches to select the egress port to forward flowlets. XMP reacts to congestion by generating ECN messages when the instantaneous queue length of any switch exceeds a threshold to inform the sender to adjust its congestion window in addition to the technique used in MPTCP.

Load Balancing: Although ECMP is a routing technique, if the output generated by the hash function used in ECMP and the incoming flows are both uniformly distributed, the distribution of flows will be uniform. In this case, ECMP may provide load balancing among equal-cost paths [135]. MPTCP, TinyFlow, and RepFlow use ECMP to determine the forwarding paths. MPTCP divides a TCP flow into several subflows using multiple address pairs (e.g., IP or port number) for the same physical source or destination servers. Hedera depends on the demand of long flows and its central scheduler to assign paths to them. Hedera may be able to determine optimal paths for invariant DCN conditions. TinyFlow uses a distributed and local approach at switches. By segmenting long flows into short flows and randomly varying the egress

port for these flows, TinyFlow provides load balancing in the network. RepFlow applies per-flow load balancing by creating another TCP connection for replicated flows (i.e., for the ones smaller than 100 Kbytes) between the sender and receiver, as MPTCP does. It requires ECMP support at the switch side. CONGA uses flowlet granularity for load balancing. It looks up the table where the congestion feedbacks are collected and selects the least congested egress port to forward a flowlet. If there are multiple egress ports that are equally uncongested, one is randomly chosen. RPS randomly selects the egress port from the equal-cost paths for each packet to balance the load on different paths.

Targeted Traffic: Hedera, TinyFlow, and XMP aim to balance the load of long flows whereas MPTCP, CONGA, and RPS aim to balance the load of all flows, in DCNs. RepFlow targets short flows only.

Implementation: Hedera uses a central controller and OpenFlow switches, while MPTCP uses the TCP/IP stack distributed throughout the DCN. TinyFlow also uses OpenFlow switches to detect and split long flows. Because RepFlow simply creates a TCP connection between the same sender-receiver pair, it does not require any modification to TCP. CONGA uses custom switching Application-Specific Integrated Circuits (ASICs) to perform its operations, such as flowlet detection, storing congestion measurement tables, load calculation for the links, generation of congestion feedbacks, and congestion marking. RPS does not require a specific implementation detail except regular commodity switches, as a more sophisticated version of RPS is already implemented on today's commodity switches [136]. XMP uses a modified version of TCP/IP stack, as does MPTCP.

In-sequence Packet Delivery: Schemes that exploit the multipath feature of DCNs forward the packets of a flow using multiple alternative paths between source and destination end hosts. However, this technique may lead to delivering packets out-of-sequence because of the latency differences (stemming from their different queueing delays) of alternative paths [49], [137]. Out-of-sequence packet delivery occurs when packets with higher sequence numbers arrive before those with lower sequence numbers, which left the sender earlier. In that case, TCP may unnecessarily retransmit delayed out-of-sequence packets [137]. Schemes that exploit the multipath feature of DCNs, such as DeTail, Hedera, MPTCP, TinyFlow, RepFlow, CONGA, RPS, and XMP, are likely to experience out-of-sequence packet delivery. Therefore, we summarize how these schemes address this issue.

DeTail disables the fast-recovery and fast-retransmit algorithms to avoid retransmissions. DeTail employs TCP NewReno for reacting to out-of-sequence packets. In addition, DeTail uses reorder buffers at end hosts for cases when out-of-sequence arrivals occur [45].

Hedera relocates a flow when the bandwidth demand of the flow exceeds 10% of the link capacity, and a new path is selected among the equal-cost paths to the destination. Because the relocation of the flow is carried out by selecting a new path, out-of-sequence packet delivery is unlikely. Out-of-sequence packets may also be experienced in MPTCP as each subflow may take a different path, and the selection of paths in MPTCP is oblivious to the path's latency. MPTCP then apportions a measure against out-of-sequence packets; each segment carries two different sequence numbers in its header. The first one is the connection-level sequence number, and the second one is a subflow-specific sequence number. By mapping these two sequence numbers, the receiver reassembles the original byte stream [48].

TinyFlow employs OpenFlow-based edge switches to randomly vary the egress port of a long flow when 10KB of data have been sent. In this way, TinyFlow segments long flows into 10KB flows. The selection of a new egress port for the 10KB segments is performed among the equal-cost paths to the destination. However, some packets of these segmented flows may be delivered out-of-sequence [49].

Although RepFlow is a multipath forwarding scheme, it does not suffer from out-of-sequence packet delivery because it employs a flow-based forwarding technique; every packet of each replicated flow follows the same path to the destination [50]. CONGA does not suffer from out-of-sequence packet delivery as long as the inter-flow gap is large enough (i.e., if the idle time between two consecutive bursts of a flow is larger than the maximum latency difference among the multiple paths to the same destination). Moreover, a flowlet-inactivity timeout parameter introduces a compromise between the number of out-of-sequence packets and the number of flowlets that exploit multiple paths. Therefore, the flowlet-inactivity timeout must be carefully selected to avoid out-of-sequence packet delivery while maintaining a satisfactory level of load balancing [51]. RPS is a packet-based forwarding scheme, which may lead to out-of-sequence packet delivery. The problem is not handled in RPS because it works under the assumption that TCP tolerates some out-of-sequence packets and the paths under RPS are expected to be equally loaded [52]. XMP may also lead to out-of-sequence packet delivery because it forwards subflows using alternative paths, as MPTCP does [53]. XMP uses the same approach as DCTCP to address this issue.

4) Schemes with Hardware Modification: We look into further details of the schemes that modify the hardware of hosts, switches, or a combination of both to run them. We discuss the features of the schemes and compare their properties. Table XI presents the working principles, the use of switch storage, host operation, and switch operation of the schemes in this category.

Working Principle: HULL is built under the assumption that long flows congest paths. It solves this issue by using a PQ to estimate the would-be occupancy if the congestion in a link increases and by pacing long flows. DAQ aims to dynamically reserve bandwidth for long flows and maintain the larger portion of it for short flows. DAQ uses weighted round-robin to reserve a minimum amount of bandwidth for long flows if the demand for bandwidth increases. CONGA aims to avoid congestion of paths by monitoring path states and by load balancing traffic. The load balancing mechanism is based on flowlets.

Switch Storage: HULL requires storage to temporarily store packets of long flows in switches as flows wait for pacing. DAQ uses three queues: two queues for storing packets of

TABLE X
COMPARISON OF MULTIPATH FORWARDING SCHEMES

Scheme	Control Plane	Reaction to Congestion	Load Balancing Method	Targeted Traffic	Implementation
MPTCP	Distributed	Relocation of subflows	ECMP + Multiple addresses from the same physical source/destination servers	All flows	Modified TCP/IP stack
Hedera	Centralized	Relocation of flows	Link-utilization-based flow re-routing + Hash-based forwarding	Long flows	OpenFlow
TinyFlow	Distributed	N/A	ECMP + Splitting long flows	Long flows	OpenFlow
RepFlow	Distributed	N/A	ECMP + Replicating short flows	Short flows	TCP (multiple sockets)
CONGA	Distributed	Selecting egress port based on congestion metric	Congestion-aware flowlet forwarding	All flows	Custom switching ASICs
RPS	Distributed	N/A	Packet spraying	All flows	Regular commodity switches
XMP	Distributed	Relocation of subflows + ECN	Multiple addresses from the same physical source/destination servers	Long flows	Modified TCP/IP stack

urgent and non-urgent short flows and one queue for packets of long flows. DAQ does not store flow states. CONGA requires storing flowlet states in (edge) switches.

Host Operation: In HULL, the sender performs pacing of long flows to alleviate or avoid congestion. DAQ senders operate as they would in a TCP/IP stack network. CONGA keeps the state of congestion at senders for different destinations and uses this information for the identification of suitable paths. In addition, most load balancing decisions are made at the senders. CONGA receivers need to temporarily store congestion (ECN) information for sending it back to senders.

Switch Operation: In HULL, switches use PQs to prevent congestion and use ECN thresholds to issue congestion notifications. In DAQ, switches perform packet scheduling and queueing of packets. Because queues in switches may have a limited size, DAQ may use data-link layer flow control to avoid packet losses. CONGA requires switches to perform several functions, including detection of flowlets and to keep routing information initiated by senders.

Table XII presents a comparison of features of the schemes in this category. We select the following features: compatibility with TCP, deadline awareness, cross-layer operation, and deployability.

Compatibility with TCP: DAQ performs most of the operation at switches and leaves end hosts mostly intact. The operations of DAQ are mostly transparent to senders and receivers; therefore, it is TCP compatible. CONGA is also reported to be compatible with TCP. The compatibility of HULL with TCP is unknown.

Deadline Awareness: Of these three schemes, DAQ is the only deadline-aware scheme. HULL and CONGA are deadline agnostic schemes.

Cross-layer Operation: All of these schemes recur to cross-layer operation. HULL performs pacing, which affects the operation of data-link layer protocols. DAQ may use flow control at the data-link layer to run a lossless fabric and identification of flows at the transport layer. CONGA uses detection of flowlets, congestion, and load balancing, and it may involve operation from data-link to application layers.

Deployability: HULL requires a hardware-based pacer at each sender's NIC. HULL switches must also be accessible for the implementation of the PQ. DAQ requires a queueing system in switches. These queues may be implemented in hardware or software. Although some switches may also be provisioned with queue structures that can be used by DAQ, the mechanism for identifying short and long flows is needed. CONGA may have medium complexity for deployment because it requires modifications at the switches. In fact, it is reported that most of the functions of the scheme are implemented in ASICs [51]. Therefore, HULL and DAQ are labeled as having high deployability compared with the medium deployability of CONGA.

TABLE XI
FEATURES OF SCHEMES BASED ON HARDWARE MODIFICATION

Scheme	Working Principle	Switch Storage	Host Operations	Switch Operation
HULL	Resolve congestion from long flows	Storing for pacing long flows	Pacing	PQ and ECN
DAQ	Weighted Round-Robin scheduling	No per flow states		Differentiated queueing
CONGA	Priority schedule, priority drop	Per flowlet states	Load balancing multipath routing	Multipath routing flowlet detection bookeeping

We summarized which surveyed scheme is proposed by academia, industry, or by a collaboration of both in Table XIII. As the table shows, most of the schemes are proposed by academia and almost half of them are proposed by both according to published information. Also, because data centers are owned and operated by private companies, information about their adopted DCN topologies information about their adopted DCN may not be available, to the best of our knowledge.

VII. TCP VERSIONS AND APPLICABILITY TO DATA CENTER NETWORKS

Although TCP is the transport protocol deployed in the Internet, it may not be a suitable protocol for DCNs for multiple reasons [18]. First, TCP is not designed to minimize FCT, whereas minimizing FCT is one of the main objectives in DCNs [9]. For instance, one of the existing versions of TCP, TCP Reno, makes the transport of flows last longer than necessary [9]. Second, the TCP incast problem may arise under the many-to-one traffic pattern, which is generated by many data center applications such as web search, web-page content composition, MapReduce, and some of the distributed file systems such as Google File System and Hadoop [18], [77], [113], [114]. Many surveyed schemes [10], [11], [40], [41], [43], [45], [47], [50], [53], [55], [56], [77] in this paper are concerned with TCP incast and aim to mitigate or alleviate it. Third, the greedy nature of long flows in data centers may cause queue buildup in switches by constantly monopolizing the buffer of the receiver switch. This queue growth may delay the transmission of short flows if short flows share the same links with long flows [18]. A similar problem called buffer pressure may arise when long flows share the link with short flows. In this case, long flows leave a small buffer space for short-flow bursts at the receiver switch [18].

A vast number of TCP congestion control algorithms, such as TCP Tahoe [126], TCP Reno [138], TCP NewReno [106], TCP Vegas [127], [139], TCP Binary Increase Congestion (BIC) control [140], TCP CUBIC [37], FAST TCP [36], and many others have been proposed in various scenarios [141]. One would wonder whether these proposed TCP congestion control algorithms may be applicable in DCNs. Although standard TCP congestion-control algorithms such as TCP Tahoe, TCP Reno, and TCP NewReno perform well for large bandwidth-delay product networks, they do not target minimizing FCT. TCP Tahoe uses the retransmission timer and the fast retransmit to detect congestion in the network and does not have the fast recovery algorithm, which may increase the FCT in case of packet loss [126]. TCP Reno and TCP NewReno depend on packet loss as the indication of congestion, and they react to congestion after the buffer of the switch fills up and starts dropping packets [106], [138]. TCP Vegas [139] and FAST TCP [36] aim to use changes in RTT for adjusting the sender's congestion window to adapt to the (incipient) congestion conditions in the network [142]. The assumption in both of these congestion-control algorithms is that the increase of RTT is only caused by the increase in queueing delay at the bottleneck switch. However, DCNs actually have very small RTTs compared with the Internet, and the increase of RTT does not apply here [77]. Moreover, queueing delay, which is the primary cause of RTT changes in a DCN, may not provide a reliable sign of congestion in data centers [10]. TCP BIC and CUBIC are designed for high speed but high latency networks (e.g., satellite communications), and they may not be suitable for DCNs, which usually have RTT values less than 250 μs in absence of queueing [10]. On the other hand, most of the TCP variants, such as TCP Tahoe, Reno, NewReno, BIC, and CUBIC may generate bursty traffic by injecting many packets back-to-back in sub-RTT time scales, which in turn may cause queueing delays, packet losses, and lower throughput [143].

VIII. CONCLUSIONS

The generated flows in a data center are coarsely classified into short and long flows, from which short flows may carry a deadline for completing their transmission. Short flows require either short flow completion times or completion before a deadline whereas long flows require being serviced at a minimum acceptable throughput. The transmission requirements of data center flows are mostly challenged by the many-to-one traffic pattern. The data center applications that generate this traffic pattern apply the Partition/Aggregate distribution model. This pattern and the coexistence of short and long flows raise challenges for the satisfaction of the performance requirements of data center flows. These challenges have prompted the design of new transport schemes. In this paper, we presented a survey of these schemes and classify them into those that are deadline aware and those that aim to minimize FCT. As one of the performance issues is throughput collapse

TABLE XII
COMPARISON OF SCHEMES BASED ON HARDWARE MODIFICATION

Scheme	Compatibility with TCP	Deadline aware	Cross-layer operation	Deployability
HULL	Not known	No	No	High
DAQ	Yes	Yes	Yes	High
CONGA	Yes	No	Yes	Medium

TABLE XIII
SCHEMES PROPOSED BY ACADEMIA/INDUSTRY

Scheme	Proposed by Academia	Proposed by Industry
DCTCP	√	√
RCP	√	√
RACS	√	
L^2DCT	√	
HULL	√	√
pFabric	√	√
DeTail	√	√
Hedera	√	
MPTCP	√	√
TinyFlow	√	
RepFlow	√	
CONGA		√
RPS	√	
XMP	√	
D^3	√	√
PDQ	√	
D^2TCP	√	√
DAQ	√	
ICTCP	√	
BCN		√
QCN	√	√
FECN	√	
E-FECN	√	

caused by TCP incast, we also addressed developed schemes to alleviate it. Herein, we have analyzed the different operation and properties of the surveyed schemes. We observed that schemes, in general, aim to reduce the flow completion time. These schemes consider network congestion to be the major hurdle for satisfying flow requirements. As a result, most schemes address network congestion in either a direct or indirect fashion. In particular, we observed that schemes that prioritize the transmission of flows according to flow size or deadline may address congestion indirectly. In these schemes, the selection of flows and their rate assignment may require fast and complex infrastructure support. Other schemes may resort to exploiting the high-level path parallelism of DCNs and perform load balancing or transmission redundancy to achieve low FCTs. We observed that these schemes may need to take special care for providing in-sequence packet forwarding. We also observed that more research is needed to evaluate the impact that out-of-sequence packet delivery has on FCT and on the number of flows that finish transmission within their deadline.

With the particular features of data center traffic, we found a new family of schemes that resort to adapting data center switches for satisfying the traffic requirements. This option presents the advantage that existing protocols may need little modification. In addition, modifications on the hardware of DCN equipment may enable precise management on high-bandwidth interconnection links.

In general, we foresee that much research is needed in designing schemes for reducing FCT considering the coexistence of long and short flows as they compete for DCN resources.

The fault-tolerance in DCNs can be listed as another open research problem. In fact, the multipath feature of the DCN may be used to improve the resilience of transport schemes against the node or link failures.

With schemes following a particular working principle, as classified herein, each of them faces particular challenges, including complexity, achievable FCT, fairness on the service for short and long flows, and implementation feasibility in software or hardware. For example, schemes that are based on solving network congestion may need to improve the efficiency of feedback mechanisms for regulating the rates in which flows are transmitted. As data centers are generally owned by a single administrator, their compatibility with TCP and other protocols of the TCP/IP suite is not strictly required because those protocols may be run within the data center only.

REFERENCES

[1] A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, and I. Stoica, "Above the clouds: A berkeley view of cloud computing," *Dept. Electrical Eng. and Comput. Sciences, University of California, Berkeley, Rep. UCB/EECS*, vol. 28, p. 13, 2009.

[2] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica *et al.*, "A view of cloud computing," *Communications of the ACM*, vol. 53, no. 4, pp. 50–58, 2010.

[3] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," in *Proc. ACM SIGCOMM*, vol. 38, no. 4. ACM, 2008, pp. 63 – 74.

[4] G. Wang and T. E. Ng, "The impact of virtualization on network performance of amazon EC2 data center," in *INFOCOM, 2010 Proceedings IEEE*. IEEE, 2010, pp. 1–9.

[5] J. Baliga, R. W. Ayre, K. Hinton, and R. S. Tucker, "Green cloud computing: Balancing energy in processing, storage, and transport," *Proc. of the IEEE*, vol. 99, no. 1, pp. 149–167, 2011.

[6] 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. ACM SIGCOMM*, vol. 39, no. 4. ACM, 2009, pp. 51–62.

[7] A. Greenberg, J. Hamilton, D. A. Maltz, and P. Patel, "The cost of a cloud: research problems in data center networks," *ACM SIGCOMM computer communication review*, vol. 39, no. 1, pp. 68–73, 2008.

[8] J. Rajahalme, A. Conta, B. Carpenter, and S. Deering, "IPv6 flow label specification," RFC 3697, March 2004. [Online]. Available: http://tools.ietf.org/html/rfc3697.

[9] N. Dukkipati and N. McKeown, "Why flow-completion time is the right metric for congestion control," *ACM SIGCOMM Computer Communication Review*, vol. 36, no. 1, pp. 59–62, 2006.

[10] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan, "Data center TCP (DCTCP)," *ACM SIGCOMM Computer Communication Review*, vol. 40, no. 4, pp. 63–74, 2010.

[11] 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.

[12] D. Meisner, C. M. Sadler, L. A. Barroso, W.-D. Weber, and T. F. Wenisch, "Power management of online data-intensive services," in *Computer Architecture (ISCA), 2011 38th Annual International Symposium on*. IEEE, 2011, pp. 319–330.

[13] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels, "Dynamo: Amazon's highly available key-value store," in *SOSP*, vol. 7, 2007, pp. 205–220.

[14] T. Hoff, "10 eBay secrets for planet wide scaling," 2009. [Online]. Available: http://goo.gl/eNjn8k.

[15] W. Vogels, "Performance and scalability," 2009. [Online]. Available: http://goo.gl/3dChwU.

[16] S. Kandula, S. Sengupta, A. Greenberg, P. Patel, and R. Chaiken, "The nature of data center traffic: measurements & analysis," in *Proc. of SIGCOMM conference on Internet measurement*. ACM, 2009, pp. 202–208.

[17] T. Benson, A. Anand, A. Akella, and M. Zhang, "Understanding data center traffic characteristics," *ACM SIGCOMM Computer Communication Review*, vol. 40, no. 1, pp. 92–99, 2010.

[18] R. P. Tahiliani, M. P. Tahiliani, and K. C. Sekaran, "TCP Variants for Data Center Networks: A Comparative Study," in *Cloud and Services Computing (ISCOS), 2012 International Symposium on*. IEEE, 2012, pp. 57–62.

[19] A. Phanishayee, E. Krevat, V. Vasudevan, D. G. Andersen, G. R. Ganger, G. A. Gibson, and S. Seshan, "Measurement and analysis of TCP throughput collapse in cluster-based storage systems," in *FAST*, vol. 8, 2008, pp. 1–14.

[20] J. Zhang, F. Ren, and C. Lin, "Modeling and understanding TCP incast in data center networks," in *INFOCOM, 2011 Proceedings IEEE*. IEEE, 2011, pp. 1377–1385.

[21] J. Dean and S. Ghemawat, "MapReduce: simplified data processing on large clusters," *Communications of the ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[22] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. Franklin, S. Shenker, and I. Stoica, "Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing," in *Proc. of the 9th USENIX conference on Networked Systems Design and Implementation*. USENIX Association, 2012.

[23] G. Malewicz, M. H. Austern, A. J. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski, "Pregel: a system for large-scale graph processing," in *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*. ACM, 2010, pp. 135–146.

[24] T. Dillon, C. Wu, and E. Chang, "Cloud computing: issues and challenges," in *Advanced Information Networking and Applications (AINA), 2010 24th IEEE International Conference on*. Ieee, 2010, pp. 27–33.

[25] T. Hoff, "Latency Is Everywhere and It Costs You - Sales How to Crush It," *High Scalability*, July 2009. [Online]. Available: http://goo.gl/8XrV3n.

[26] H. Kopetz, *Real-time systems: design principles for distributed embedded applications*. Springer, 2011.

[27] 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," *Proc. ACM SIGCOMM*, vol. 39, no. 4, pp. 63–74, 2009.

[28] C. Guo, H. Wu, K. Tan, L. Shi, Y. Zhang, and S. Lu, "DCell: a scalable and fault-tolerant network structure for data centers," in *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 4. ACM, 2008, pp. 75–86.

[29] J. Hamilton, "An architecture for modular data centers," *arXiv preprint cs/0612110*, 2006.

[30] K. V. Vishwanath, A. Greenberg, and D. A. Reed, "Modular data centers: how to design them?" in *Proceedings of the 1st ACM workshop on Large-Scale system and application performance*. ACM, 2009, pp. 3–10.

[31] M. M. Waldrop, "Data center in a box," *Scientific American*, vol. 297, no. 2, pp. 90–93, 2007.

[32] "SGI ICE Cube Air," December 2010. [Online]. Available: http://goo.gl/fY93ae.

[33] Y. Zhang and M. Ahmed, "A control theoretic analysis of XCP," in *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE*, vol. 4. IEEE, 2005, pp. 2831–2835.

[34] S. Bhandarkar, S. Jain, and A. Reddy, "LTCP: improving the performance of TCP in highspeed networks," *ACM SIGCOMM Computer Communication Review*, vol. 36, no. 1, pp. 41–50, 2006.

[35] S. Floyd, "Highspeed TCP for large congestion windows," in *RFC 3649*, 2003. [Online]. Available: https://tools.ietf.org/html/rfc3649.

[36] C. Jin, D. X. Wei, and S. H. Low, "FAST TCP: motivation, architecture, algorithms, performance," in *INFOCOM 2004. Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies*, vol. 4. IEEE, 2004, pp. 2490–2501.

[37] S. Ha, I. Rhee, and L. Xu, "CUBIC: a new TCP-friendly high-speed TCP variant," *ACM SIGOPS Operating Systems Review*, vol. 42, no. 5, pp. 64–74, 2008.

[38] L. Xu, K. Harfoush, and I. Rhee, "Binary increase congestion control (BIC) for fast long-distance networks," in *INFOCOM 2004. Twenty-third Annual Joint Conference of the IEEE Computer and Communications Societies*, vol. 4. IEEE, 2004, pp. 2514–2524.

[39] N. Dukkipati, M. Kobayashi, R. Zhang-Shen, and N. McKeown, "Processor sharing flows in the Internet," in *Quality of Service–IWQoS 2005*. Springer, 2005, pp. 271–285.

[40] A. Munir, I. Qazi, and S. Bin Qaisar, "On achieving low latency in data centers," in *Communications (ICC), 2013 IEEE International Conference on*, June 2013, pp. 3721–3725.

[41] A. Munir, I. A. Qazi, Z. A. Uzmi, A. Mushtaq, S. N. Ismail, M. S. Iqbal, B. Khan, and Y. Li, "Minimizing flow completion times in data centers," in *Proceedings of IEEE INFOCOM*, 2013.

[42] 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. of NSDI*, 2012.

[43] M. Alizadeh, S. Yang, S. Katti, N. McKeown, B. Prabhakar, and S. Shenker, "Deconstructing datacenter packet transport," in *Proceedings of the 11th ACM Workshop on Hot Topics in Networks*. ACM, 2012, pp. 133–138.

[44] M. Alizadeh, S. Yang, M. Sharif, S. Katti, N. McKeown, B. Prabhakar, and S. Shenker, "pFabric: Minimal near-optimal datacenter transport," ACM, 2013.

[45] D. Zats, T. Das, P. Mohan, D. Borthakur, and R. Katz, "DeTail: Reducing the flow completion time tail in datacenter networks," *ACM SIGCOMM Computer Communication Review*, vol. 42, no. 4, pp. 139–150, 2012.

[46] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, "Hedera: Dynamic flow scheduling for data center networks," in *NSDI*, vol. 10, 2010.

[47] C. Raiciu, S. Barre, C. Pluntke, A. Greenhalgh, D. Wischik, and M. Handley, "Improving datacenter performance and robustness with multipath TCP," in *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 4. ACM, 2011, pp. 266–277.

[48] A. Ford, C. Raiciu, M. Handley, and O. Bonaventure, "TCP extensions for multipath operation with multiple addresses," RFC 6824, January 2013. [Online]. Available: https://tools.ietf.org/html/rfc6824.

[49] H. Xu and B. Li, "TinyFlow: Breaking elephants down into mice in data center networks," *IEEE LANMAN*, 2014.

[50] ——, "RepFlow: Minimizing flow completion times with replicated flows in data centers," in *INFOCOM, 2014 Proceedings IEEE*, April 2014, pp. 1581–1589.

[51] M. Alizadeh, T. Edsall, S. Dharmapurikar, R. Vaidyanathan, K. Chu, A. Fingerhut, V. The Lam, F. Matus, R. Pan, N. Yadav, and G. Varghese, "CONGA: Distributed congestion-aware load balancing for datacenters," in *Proc. of SIGCOMM*. ACM, 2014.

[52] A. Dixit, P. Prakash, Y. Hu, and R. Kompella, "On the impact of packet spraying in data center networks," in *INFOCOM, 2013 Proceedings IEEE*, April 2013, pp. 2130–2138.

[53] Y. Cao, M. Xu, X. Fu, and E. Dong, "Explicit multipath congestion control for data center networks," in *Proceedings of the ninth ACM conference on Emerging networking experiments and technologies*. ACM, 2013, pp. 73–84.

[54] C.-Y. Hong, M. Caesar, and P. Godfrey, "Finishing flows quickly with preemptive scheduling," *ACM SIGCOMM Computer Communication Review*, vol. 42, no. 4, pp. 127–138, 2012.

[55] B. Vamanan, J. Hasan, and T. Vijaykumar, "Deadline-aware datacenter TCP (D2TCP)," in *Proceedings of the ACM SIGCOMM 2012 conference on Applications, technologies, architectures, and protocols for computer communication*. ACM, 2012, pp. 115–126.

[56] C. Ding and R. Rojas-Cessa, "DAQ: Deadline-Aware Queue scheme for scheduling service flows in data centers," in *International Conference on Communications*. IEEE, 2014, p. 6.

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

[58] Y. Zhang and N. Ansari, "On architecture design, congestion notification, TCP incast and power consumption in data centers," *Communications Surveys & Tutorials, IEEE*, vol. 15, no. 1, pp. 39–64, 2013.

[59] K. Bilal, S. U. Khan, L. Zhang, H. Li, K. Hayat, S. A. Madani, N. Min-Allah, L. Wang, D. Chen, M. Iqbal *et al.*, "Quantitative comparisons of the state-of-the-art data center architectures," *Concurrency and Computation: Practice and Experience*, vol. 25, no. 12, pp. 1771–1783, 2013.

[60] T. Wang, Z. Su, Y. Xia, and M. Hamdi, "Rethinking the data center networking: Architecture, network protocols, and resource sharing," *Access, IEEE*, vol. 2, pp. 1481–1496, 2014.

[61] K. Bilal, S. U. R. Malik, O. Khalid, A. Hameed, E. Alvarez, V. Wijaysekara, R. Irfan, S. Shrestha, D. Dwivedy, M. Ali *et al.*, "A taxonomy and survey on green data center networks," *Future Generation Computer Systems*, vol. 36, pp. 189–208, 2014.

[62] "Data Center Multi-Tier Model Design," CISCO Data Center Infrastructure 2.5 Design Guide, December 2007. [Online]. Available: http://goo.gl/HRZPxC.

[63] P. Costa, A. Donnelly, G. Oshea, and A. Rowstron, "CamCube: a key-based data center," Technical Report MSR TR-2010-74, Microsoft Research, Tech. Rep., 2010.

[64] H. Abu-Libdeh, P. Costa, A. Rowstron, G. O'Shea, and A. Donnelly, "Symbiotic routing in future data centers," *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 4, pp. 51–62, 2011.

[65] N. Farrington, E. Rubow, and A. Vahdat, "Data center switch architecture in the age of merchant silicon," in *High Performance Interconnects, 2009. HOTI 2009. 17th IEEE Symposium on*. IEEE, 2009, pp. 93–102.

[66] H. Wu, G. Lu, D. Li, C. Guo, and Y. Zhang, "MDCube: a high performance network structure for modular data center interconnection," in *Proceedings of the 5th international conference on Emerging networking experiments and technologies*. ACM, 2009, pp. 25–36.

[67] D. Lin, Y. Liu, M. Hamdi, and J. Muppala, "Hyper-BCube: A scalable data center network," in *Communications (ICC), 2012 IEEE International Conference on*. IEEE, 2012, pp. 2918–2923.

[68] A. Singla, C.-Y. Hong, L. Popa, and P. B. Godfrey, "Jellyfish: Networking Data Centers Randomly," in *NSDI*, vol. 12, 2012, pp. 17–17.

[69] R. Niranjan Mysore, A. Pamboris, N. Farrington, N. Huang, P. Miri, S. Radhakrishnan, V. Subramanya, and A. Vahdat, "PortLand: a scalable fault-tolerant layer 2 data center network fabric," in *ACM SIGCOMM Computer Communication Review*, vol. 39, no. 4. ACM, 2009, pp. 39–50.

[70] 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 *Communications (ICC), 2014 IEEE International Conference on*. IEEE, 2014, pp. 4005–4010.

[71] D. Li, C. Guo, H. Wu, K. Tan, Y. Zhang, and S. Lu, "FiConn: Using backup port for server interconnection in data centers," in *INFOCOM 2009, IEEE*. IEEE, 2009, pp. 2276–2285.

[72] D. Lin, Y. Liu, M. Hamdi, and J. Muppala, "FlatNet: Towards a flatter data center network," in *Global Communications Conference (GLOBECOM), 2012 IEEE*. IEEE, 2012, pp. 2499–2504.

[73] K. Chen, A. Singla, A. Singh, K. Ramachandran, L. Xu, Y. Zhang, X. Wen, and Y. Chen, "OSA: an Optical Switching Architecture

for data center networks with unprecedented flexibility," *IEEE/ACM Transactions on Networking (TON)*, vol. 22, no. 2, pp. 498–511, 2014.

[74] G. Wang, D. G. Andersen, M. Kaminsky, K. Papagiannaki, T. Ng, M. Kozuch, and M. Ryan, "c-Through: Part-time optics in data centers," *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 4, pp. 327–338, 2011.

[75] 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," *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 4, pp. 339–350, 2011.

[76] J.-Y. Shin, B. Wong, and E. G. Sirer, "Small-World datacenters," in *Proceedings of the 2nd ACM Symposium on Cloud Computing*. ACM, 2011, p. 2.

[77] H. Wu, Z. Feng, C. Guo, and Y. Zhang, "ICTCP: Incast Congestion Control for TCP in data center networks," *IEEE/ACM Transactions on Networking*, vol. 21, no. 2, p. 345, 2013.

[78] T. Benson, A. Akella, and D. A. Maltz, "Network traffic characteristics of data centers in the wild," in *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement*, 2010, pp. 267–280.

[79] T. Benson, A. Anand, A. Akella, and M. Zhang, "Microte: fine grained traffic engineering for data centers," in *Proceedings of the Seventh Conference on emerging Networking EXperiments and Technologies*. ACM, 2011, p. 8.

[80] Y. Chen, S. Jain, V. K. Adhikari, Z.-L. Zhang, and K. Xu, "A first look at inter-data center traffic characteristics via Yahoo! datasets," in *INFOCOM, 2011 Proceedings IEEE*. IEEE, 2011, pp. 1620–1628.

[81] N. Farrington and A. Andreyev, "Facebook's data center network architecture," in *Optical Interconnects Conference, 2013 IEEE*, May 2013, pp. 49–50.

[82] K. Chen, C. Hu, X. Zhang, K. Zheng, Y. Chen, and A. V. Vasilakos, "Survey on routing in data centers: insights and future directions," *Network, IEEE*, vol. 25, no. 4, pp. 6–10, 2011.

[83] M. C. Sriram Sankar, Soren Lassen. (2013, March) Under the Hood: Building out the infrastructure for Graph Search. [Online]. Available: http://goo.gl/aI44Y4.

[84] H. Karloff, S. Suri, and S. Vassilvitskii, "A model of computation for MapReduce," in *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms*. Society for Industrial and Applied Mathematics, 2010, pp. 938–948.

[85] S. S. R. Maripalli and G. Abirami, "Congestion Control for TCP in Data-Center Networks," *International Journal of Computer Science and Information Technologies (IJCSIT)*, vol. Vol. 5 (2), pp. 1903–1907, 2014.

[86] R. S. Sujal Das. (2012, April) Broadcom Smart-Buffer Technology in Data Center Switches for Cost-Effective Performance Scaling of Cloud Applications. [Online]. Available: https://www.broadcom.com/collateral/etp/SBT-ETP100.pdf

[87] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The Google file system," in *ACM SIGOPS Operating Systems Review*, vol. 37, no. 5. ACM, 2003, pp. 29–43.

[88] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The Hadoop distributed file system," in *Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on*. IEEE, 2010, pp. 1–10.

[89] K. Church, A. G. Greenberg, and J. R. Hamilton, "On delivering embarrassingly distributed cloud services," in *HotNets*, 2008, pp. 55–60.

[90] M. Nelson, B.-H. Lim, G. Hutchins *et al.*, "Fast transparent migration for virtual machines." in *USENIX Annual Technical Conference, General Track*, 2005, pp. 391–394.

[91] VMware vMotion. [Online]. Available: http://goo.gl/pn4Tle.

[92] H. Chen, H. Kang, G. Jiang, and Y. Zhang, "Coordinating virtual machine migrations in enterprise data centers and clouds." [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.417.9732&rep=rep1&type=pdf

[93] I. Takouna, R. Rojas-Cessa, K. Sachs, and C. Meinel, "Communication-aware and energy-efficient scheduling for parallel applications in virtualized data centers," in *Utility and Cloud Computing (UCC), 2013 IEEE/ACM 6th International Conference on*. IEEE, 2013, pp. 251–255.

[94] M. Alizadeh, A. Javanmard, and B. Prabhakar, "Analysis of DCTCP: stability, convergence, and fairness," in *Proceedings of the ACM SIGMETRICS joint international conference on Measurement and modeling of computer systems*. ACM, 2011, pp. 73–84.

[95] K. Ramakrishnan, S. Floyd, D. Black *et al.*, "The addition of Explicit Congestion Notification (ECN) to IP," RFC 3168, September 2001. [Online]. Available: http://tools.ietf.org/html/rfc3168.

[96] E. G. Coffman Jr, R. R. Muntz, and H. Trotter, "Waiting time distributions for processor-sharing systems," *Journal of the ACM (JACM)*, vol. 17, no. 1, pp. 123–130, 1970.

[97] L. E. Schrage and L. W. Miller, "The queue M/G/1 with the shortest remaining processing time discipline," *Operations Research*, vol. 14, no. 4, pp. 670–684, 1966.

[98] L. Schrage, "A proof of the optimality of the shortest remaining processing time discipline," *Operations Research*, vol. 16, no. 3, pp. 687–690, 1968.

[99] B. N and H.-B. M, "Analysis of SRPT scheduling: Investigating unfairness," in *Proceedings of the 2001 ACM SIGMETRICS*, ser. SIGMETRICS '01, 2001, pp. 279–290.

[100] I. A. Rai, G. Urvoy-Keller, and E. W. Biersack, "Analysis of LAS scheduling for job size distributions with high variance," *ACM SIGMETRICS Performance Evaluation Review*, vol. 31, no. 1, pp. 218–228, 2003.

[101] V. Sivaraman, H. A. ElGindy, D. Moreland, and D. Ostry, "Packet pacing in short buffer optical packet switched networks," in *INFOCOM*, 2006.

[102] "CISCO Data center bridging," 2010. [Online]. Available: http://goo.gl/imCxqh.

[103] "Priority flow control: Build reliable layer 2 infrastructure." [Online]. Available: http://goo.gl/CwGkkD.

[104] "CISCO Nexus 5000 series architecture." [Online]. Available: http://goo.gl/DhZA3A.

[105] N. McKeown, "White paper: A fast switched backplane for a gigabit switched router." [Online]. Available: http://goo.gl/oC1q8c.

[106] T. Henderson, S. Floyd, A. Gurtov, and Y. Nishida, "The NewReno modification to TCP's fast recovery algorithm," *RFC 6582*, 2012. [Online]. Available: https://tools.ietf.org/html/rfc6582.

[107] E. L. Hahne, "Round-robin scheduling for max-min fairness in data networks," *Selected Areas in Communications, IEEE Journal on*, vol. 9, no. 7, pp. 1024–1039, 1991.

[108] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

[109] C. D. G. S. Kirkpatrick and M. P. Vecchi, "Optimization by simulated annealing," *SCIENCE*, vol. 220, no. 4598, pp. 671–680, 1983.

[110] S. Kandula, D. Katabi, S. Sinha, and A. Berger, "Dynamic load balancing without packet reordering," *SIGCOMM Comput. Commun. Rev.*, vol. 37, no. 2, pp. 51–62, Mar. 2007.

[111] M. Allman, V. Paxson, and E. Blanton, "TCP Congestion Control," RFC 5681, September, Tech. Rep., 2009. [Online]. Available: https://tools.ietf.org/html/rfc5681.

[112] N. Chrysos, F. Neeser, M. Gusat, C. Minkenberg, W. Denzel, and C. Basso, "All routes to efficient datacenter fabrics," in *Proceedings of the 8th International Workshop on Interconnection Network Architecture: On-Chip, Multi-Chip*. ACM, 2014, p. 4.

[113] Y. Chen, R. Griffith, J. Liu, R. H. Katz, and A. D. Joseph, "Understanding TCP incast throughput collapse in datacenter networks," in *Proceedings of the 1st ACM workshop on Research on enterprise networking*. ACM, 2009, pp. 73–82.

[114] V. Vasudevan, A. Phanishayee, H. Shah, E. Krevat, D. G. Andersen, G. R. Ganger, G. A. Gibson, and B. Mueller, "Safe and effective fine-grained TCP retransmissions for datacenter communication," in *ACM SIGCOMM Computer Communication Review*, vol. 39, no. 4. ACM, 2009, pp. 303–314.

[115] D. Bergamasco and R. Pan, "Backward congestion notification version 2.0," in *IEEE 802.1 Meeting*, 2005.

[116] D. Bergamasco, "Ethernet Congestion Manager," in *IEEE 802.1Qau Meeting*, 2007.

[117] J. Jiang and R. Jain, "Analysis of Backward Congestion Notification (BCN) for Ethernet in datacenter applications," in *INFOCOM 2007. 26th IEEE International Conference on Computer Communications. IEEE*. IEEE, 2007, pp. 2456–2460.

[118] M. Alizadeh, B. Atikoglu, A. Kabbani, A. Lakshmikantha, R. Pan, B. Prabhakar, and M. Seaman, "Data center transport mechanisms: Congestion control theory and IEEE standardization," in *Communication, Control, and Computing, 2008 46th Annual Allerton Conference on*. IEEE, 2008, pp. 1270 – 1277.

[119] J. Jiang, R. Jain, and C. So-In, "An explicit rate control framework for lossless Ethernet operation," in *Communications, 2008. ICC'08. IEEE International Conference on*. IEEE, 2008, pp. 5914–5918.

[120] C. So-In, R. Jain, and J. Jiang, "Enhanced Forward Explicit Congestion Notification (E-FECN) scheme for datacenter Ethernet networks," in *Performance Evaluation of Computer and Telecommunication Systems,*

2008. SPECTS 2008. International Symposium on. IEEE, 2008, pp. 542–546.

[121] O. Feuser and A. Wenzel, "On the effects of the IEEE 802.3x flow control in full-duplex Ethernet LANs," in *Local Computer Networks, 1999. LCN'99. Conference on*. IEEE, 1999, pp. 160–161.

[122] Internet Engineering Task Force (IETF). [Online]. Available: http://www.ietf.org/.

[123] "Data Center Bridging Task Group." [Online]. Available: http://www.ieee802.org/1/pages/dcbridges.html.

[124] A. Kabbani, M. Alizadeh, M. Yasuda, R. Pan, and B. Prabhakar, "AF-QCN: Approximate fairness with quantized congestion notification for multi-tenanted data centers," in *High Performance Interconnects (HOTI), 2010 IEEE 18th Annual Symposium on*. IEEE, 2010, pp. 58–65.

[125] Y. Zhang and N. Ansari, "On mitigating TCP incast in data center networks," in *INFOCOM, 2011 Proceedings IEEE*. IEEE, 2011, pp. 51–55.

[126] V. Jacobson, "Congestion avoidance and control," in *ACM SIGCOMM Computer Communication Review*, vol. 18, no. 4. ACM, 1988, pp. 314–329.

[127] L. S. Brakmo and L. L. Peterson, "TCP Vegas: End to end congestion avoidance on a global internet," *Selected Areas in Communications, IEEE Journal on*, vol. 13, no. 8, pp. 1465–1480, 1995.

[128] D. Crisan, R. Birke, G. Cressier, C. Minkenberg, and M. Gusat, "Got loss? get zOVN!" in *Proc. of the ACM SIGCOMM 2013 Conference on SIGCOMM*, ser. SIGCOMM '13, 2013, pp. 423–434.

[129] G. F. Pfister and V. A. Norton, "Hot spot contention and combining in multistage interconnection networks," *IEEE Transactions on Computers*, vol. 34, no. 10, pp. 943–948, 1985.

[130] "WRED Explicit Congestion Notification." [Online]. Available: http://goo.gl/3T15fU.

[131] C. Joe-Wong, S. Sen, T. Lan, and M. Chiang, "Multi-resource allocation: Fairness-efficiency tradeoffs in a unifying framework," in *IEEE Proc. INFOCOM*, 2012, pp. 1206–1214.

[132] L. Popa, G. Kumar, M. Chowdhury, A. Krishnamurthy, S. Ratnasamy, and I. Stoica, "FairCloud: sharing the network in cloud computing," in *Proceedings of the ACM SIGCOMM 2012 conference on Applications, technologies, architectures, and protocols for computer communication*. ACM, 2012, pp. 187–198.

[133] A. Ford, C. Raiciu, M. Handley, S. Barre, and J. Iyengar, "Architectural guidelines for multipath TCP development," *RFC6182*, March 2011. [Online]. Available: https://tools.ietf.org/html/rfc6182

[134] "OpenFlow." [Online]. Available: https://www.opennetworking.org.

[135] C. Hopps, "Analysis of an equal-cost multi-path algorithm," RFC 2992, November 2000. [Online]. Available: https://tools.ietf.org/html/rfc2992.

[136] *CISCO, Per packet load balancing*. [Online]. Available: http://goo.gl/pYnsGS.

[137] D. Thaler and C. Hopps, "Multipath issues in unicast and multicast next-hop selection," RFC 2991, 2000. [Online]. Available: http://tools.ietf.org/html/rfc2991.

[138] G. R. Wright and W. R. Stevens, *TCP/IP Illustrated*. Addison-Wesley Professional, 1995, vol. 2.

[139] L. S. Brakmo, S. W. O'Malley, and L. L. Peterson, *TCP Vegas: New techniques for congestion detection and avoidance*. ACM, 1994, vol. 24, no. 4.

[140] V. Jacobson and R. Braden, "TCP extensions for long-delay paths," *ISI*, 1988.

[141] J. Wang, J. Wen, J. Zhang, and Y. Han, "TCP-FIT: An improved TCP congestion control algorithm and its performance," in *INFOCOM, 2011 Proceedings IEEE*. IEEE, 2011, pp. 2894–2902.

[142] J. Martin, A. Nilsson, and I. Rhee, "Delay-based congestion avoidance for TCP," *IEEE/ACM Transactions on Networking (TON)*, vol. 11, no. 3, pp. 356–369, 2003.

[143] M. Ghobadi and Y. Ganjali, "TCP pacing in data center networks," in *High-Performance Interconnects (HOTI), 2013 IEEE 21st Annual Symposium on*. IEEE, 2013, pp. 25–32.