IEEE Access
Multidisciplinary : Rapid Review : Open Access Journal

Received November 21, 2018, accepted December 21, 2018, date of publication January 7, 2019, date of current version January 29, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2891264

Incast Mitigation in a Data Center Storage Cluster Through a Dynamic Fair-Share Buffer Policy

YAWAR ABBAS BANGASH[1], TAUSEEF RANA[1], HAIDER ABBAS[2], (Senior Member, IEEE), MUHAMMAD ALI IMRAN[3], (Senior Member, IEEE), AND ADNAN AHMED KHAN[1]

[1]Department of Computer Software Engineering, National University of Sciences and Technology, Islamabad 44000, Pakistan
[2]Department of Information Security, National University of Sciences and Technology, Islamabad 44000, Pakistan
[3]School of Engineering, University of Glasgow, Glasgow G12 8QQ, U.K.

Corresponding author: Mohammad Ali Imran (muhammad.imran@glasgow.ac.uk)

This work was supported by EPSRC Global Challenges Research Fund—the DARE project—under Grant EP/P028764/1.

ABSTRACT Incast is a phenomenon when multiple devices interact with only one device at a given time. Multiple storage senders overflow either the switch buffer or the single-receiver memory. This pattern causes all concurrent-senders to stop and wait for buffer/memory availability, and leads to a packet loss and retransmission—resulting in a huge latency. We present a software-defined technique tackling the many-to-one communication pattern—Incast—in a data center storage cluster. Our proposed method decouples the default TCP windowing mechanism from all storage servers, and delegates it to the software-defined storage controller. The proposed method removes the TCP saw-tooth behavior, provides a global flow awareness, and implements the dynamic fair-share buffer policy for end-to-end I/O path. It considers all I/O stages (applications, device drivers, NICs, switches/routers, file systems, I/O schedulers, main memory, and physical disks) while achieving the maximum I/O throughput. The policy, which is part of the proposed method, allocates fair-share bandwidth utilization for all storage servers. Priority queues are incorporated to handle the most important data flows. In addition, the proposed method provides better manageability and maintainability compared with traditional storage networks, where data plane and control plane reside in the same device.

INDEX TERMS Incast, software-defined storage, I/O throughput, dynamic fair-share buffer, end-to-end I/O path, fair-share BW utilization.

I. INTRODUCTION

A Software Defined Data Center (SDDC) consists of a Software Defined Storage (SDS), a Software Defined Networking (SDN), and a Software Defined Computing (SDC). In SDDC, multiple resources are distributed across different servers in to improve access performance for various categories of data and tasks. In such a paradigm, a cluster of storage servers is networked together. Data blocks are stripped over these servers for concurrent access to facilitate small and big files, related and un-related data, concurrent and random data, web searches, maps, and data warehousing. The purpose is to deliver an application level throughput. When multiple storage servers request a single storage server for a short time, many data flows converge on the same interface of a switch/server overwhelm either the switch/server shared-memory or the allowed interface's buffer capacity. This many-to-one communication pattern leads to a phenomenon known as **Incast**. An Incast model with a single switch and a single I/O request in a storage cluster is shown in Figure 1. In this figure, the Meta Data Server (MDS) manages and stores all metadata, and provides information about the requested file. The file is stripped over N storage servers (HDD, SSD), and all stripes are accessed via a synchronized read. These stripes are returned directly to the requesting I/O machine/server without routing through MDS.

In the real world, the situation is more complex: multiple I/O requests from multiple servers also lead to Incast in a storage cluster. This situation is depicted in Figure 2. Even the simplest situation consists of multiple switches with multiple concurrent I/O accesses. For the single-server scenario, the traffic is from the storage servers to the requesting machine. For the multiple-servers scenario, the condition may apply for traffic in both directions. Providing the general idea about Incast, we do not include multiple switches' figure.

Incast was first studied by Nagle *et al.* [1] where they found throughput collapse for an increased number of concurrent

2169-3536 © 2019 IEEE. Translations and content mining are permitted for academic research only.
Personal use is also permitted, but republication/redistribution requires IEEE permission.
See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.

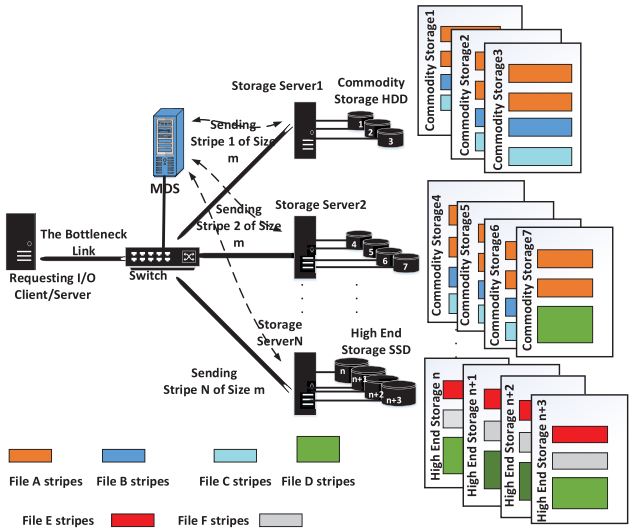

FIGURE 1. A general Incast model for a single switch: For every file access, the client/server requests the MDS to provide metadata (stripe ID and location).

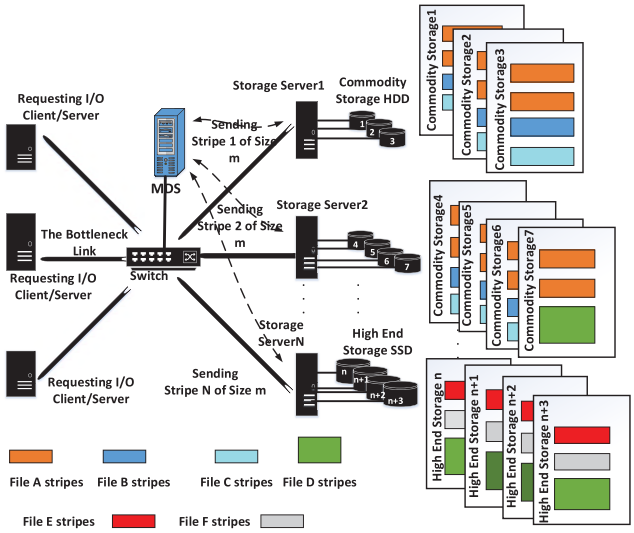

FIGURE 2. A single switch with multiple concurrent I/O requests from multiple clients.

storage senders. Switch buffers exhaust when more traffic is forwarded to a port than it can send/receive. Reasons include: ingress and egress port speed mismatch, multiple inputs to a single output port, half-duplex collision on an out-put port, the complex interconnection in data centers equipment (switches, servers, links, etc.), and the deployed communication protocol. The main problem occurs in a storage cluster, where multiple storage servers send data and cannot send more data until all parallel threads completed [2]. The application level throughput decreases far below the available bandwidth (BW) when the synchronized concurrent senders increase. In addition, in a parallel file system, when a single I/O request is issued for data stripped over multiple file servers, I/O request has to wait for all aggregator storage nodes to complete [3]. This intra request synchronization

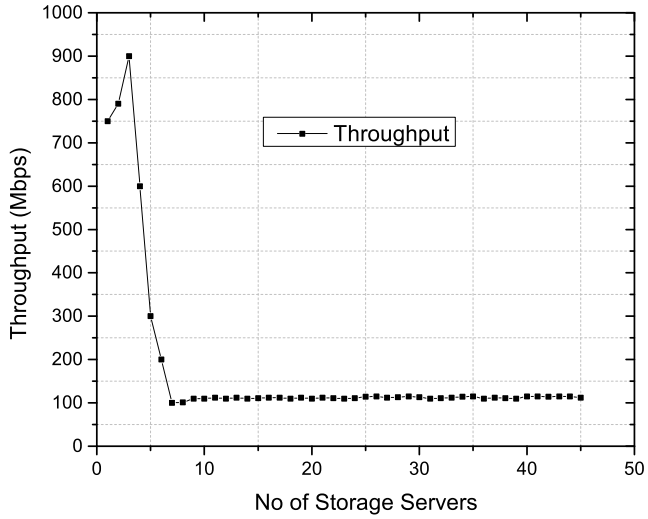

FIGURE 3. Throughput collapse in a data center storage cluster [5]: As the number of storage servers increases and, hence their I/O requests, the I/O throughput collapse occurs.

leads to Incast in storage. A brief overview of Incast problem can be found at [4]. A common throughput collapse behavior is depicted in Figure 3, where Phanishayee *et al.* [5] observed that when the number of concurrent sending storage servers exceeds seven, the overall throughput collapse starts. The default TCP fails to handle multiple concurrent storage servers. If the switch's buffer capacity is big enough to accommodate the traffic burst, throughput collapse will not occur. However, in practical situation, big switch buffers are expensive and causes delay in the network. For multiple concurrent senders, the throughput collapse stays the same irrespective of the data transfer size (small data size and big data size).

The throughput collapse results in an increased latency, and causes harm to any network-based business activities. Latency is an important metric in a computer network, specially in a delay sensitive distributed storage application [6]. The main aim is to improve the end-user experience. In emergency and disaster scenarios such as earthquake, tsunami, and terrorism, a container based data center requires as minimum as possible delay to provide service level agreement. Mayer [7] presented at Web 2.0 conference, that Google's traffic is reduced by 20% due to 500ms increase in latency. Stefanov [8] in YSlow (a web page analyzer based on Yahoo rules) stated that in Yahoo, an extra 400ms reduced the traffic by 9%, and every 100ms latency costs 1% in business revenue to Amazon. These results suggest to find a solution to minimize latency as much as possible.

McKeown *et al.* [9] presented SDN where a control plane is decoupled from a data plane, and provided a centralized policy to manage and control all networking operations (packet drop, modify, forward, update). In SDN, the control plane is moved out of a commodity switch and housed in the main controller. According to our best knowledge, a must-read comprehensive survey about SDN can be found at [10].

A simple and a robust algorithm design can cope with Incast rather than to modify the whole protocol stack. Such an algorithm can manage buffer allocation dynamically, and thus, can make an over-sized buffer void [11]. Inspired by SDN decoupled architecture, we propose a Dynamic Fair-share Buffer Policy (DyFaShBuP) for End-to-End I/O path. DyFaShBuP handles Incast in a data center storage cluster. It decouples the windowing mechanism from TCP, and delegates it to the SDS controller. An OpenFlow enabled switch collects storage servers statistics, e.g., IP addresses, MAC addresses, ports speed, buffer capacity, and window size (*WS*), and forwards these collected information to the controller. According to the network stats and number of concurrent sending storage servers, the controller re-defines and updates the policy, which is further distributed to the concurrent sending storage servers. Upon receiving the updated policy, all storage senders modify and adjust *WS* according to the available buffer capacity. The centralized controlled approach achieves high burst data-flows, provides a minimum latency, allocates a dynamic fair-share buffer (*WS*) and BW for end-to-end I/O path, and guarantees application level throughput.

Our key contributions are as follows: This is a comprehensive study to dissect Incast in many aspects, ranges from hardware architecture (link, switch, NIC) to software implementation (protocol, OS, file system). We present a dynamic fair-share buffer policy for end-to-end I/O path. This policy guarantees the fair-share buffer and I/O throughput, and does not limit the number of concurrent storage servers, and, thus it achieves scalability. We propose a novel software-defined inspired Incast tackling technique in a data center storage cluster. The default TCP windowing mechanism (windows size and flow control) is de-coupled from all storage servers and delegated to the SDS controller. The proposed method does not incorporate sender and receiver cooperation preventing Incast, but a centralized controller is responsible to cope with a momentary data-burst. The simulation results confirm that a large number of concurrent storage servers do not lead to Incast, while achieving the maximum I/O throughput. The rest of the paper is organized as follows. Section II discusses the background, Section III overviews the related work about Incast; Section IV discusses Incast anatomy. Section V is about the proposed method. Section VI covers Incast scenarios, performance analysis, and discussion, while Section VII concludes the paper.

II. BACKGROUND

SDS is an emerging storage paradigm that provides a centralized, de-coupled, and a hierarchical structure to facilitate scalability, redundancy, performance, security, manageability, and maintainability. According to the IDC report (2013), software based storage services will slowly replace traditional storage services for every data center. The decentralized approach in traditional storage systems suffers from reconfiguration of application programs, long time-consuming planning and up-gradation, maintenance activities, and storage scalability issues. In SDS paradigm, examining the I/O data path for different processes used by different tenants helps identifying the possible potential bottleneck [12].

In SDS, the intelligent and centralized server-address management ease the processing overhead on end hosts (storage servers). However, a tremendous communication with the SDS controller can lead to latency. A proper method is needed to handle the controller-switch latency. In addition, in a data center storage paradigm, different traffic workloads exist (e.g., short flow, long flow). TCP fails to meet these requirements (the ability to support both short and long flows) [13]. Hundreds of thousands of Virtual Machines (VMs) hide massive details of network topology. Managing a frequent communication system is difficult without the software-defined concept. Proprietary based storage solutions are rigid to go beyond their products and, thus, lack in innovations and flexibility. A modern storage platform should address these issues: topology deployment, un-manageable cable connection, proper cooling system, control and management, scalable routing for short and long flows, and performance and reliability. SDS controller addresses application level demands without hardware changes. SDS solutions are software based where a data plane and a control plane are de-coupled.

In our proposed method, key advantages of SDS controller are as follow: new features addition on the go, reduce complexity, efficient storage resource utilization, global flow visibility, managed modularity, dynamic fair-share buffer allocation for end-to-end I/O path, high throughput guarantee, tolerance for short-term congestion, and decoupling the data plane from the control plane. In addition, DCTCP [17] and ICTCP [18] are distributed solutions (based on sender or receiver coordination), while our proposed method is based on centralized control mechanism. SDN paradigm can implement new policy and re-program end hosts [14].

For a reliable data center, we should not only scale the storage and consider all its factors, but we should consider the whole system. From external point of view, a data center is an independent domain, but from the data center point of view, it is a complete dependent system. All communicating entities depend on each other somehow. A well-managed data center should address all system's parameters such as switch fabrics and port design, controller design, memory structure, distributed file system, network system, I/O paths, and allocated BW. Consequently, the storage system will prevent Incast. However, if a single control plane receives plenteous packets and flow requests, it can reach to a state called control plane saturation [15]. The underlined protocol should address control plane saturation.

III. RELATED WORK

A. INCAST IN TRADITIONAL STORAGE SYSTEMS

Nagle *et al.* [1] and Phanishayee *et al.* [5] reported the first work related to Incast in a distributed storage cluster. Nagle *et al.* [1] observed a throughput collapse in Panasas ActiveScale storage cluster when the number of concurrent

TABLE 1. Different techniques in the literature coping Incast.

Incast handling strategy	Description
Ethernet based [5]	Ethernet Flow Control based Incast mitigation.
Window based [17], [18]	Buffer modification at the sending and receiving ends.
RTO based [19]	The reduction of RTO to tackle Incast.
MTU based [20]	Incast mitigation through MTU shrinking.
RTT based [21]	RTT is a holistic signal compared to ECN to mitigate Incast.
FQCN based [22]	The Incast mitigation through a fair Quantized Congestion Notification (FQCN) to improve multiple flows' fairness over a shared bottleneck link.
Switch based [23], [24]	The Incast mitigation through a switch and ECN, and SDN switch.
Number of concurrent senders [16]	Incast mitigation via limiting the number of concurrent senders.
Controller based [25], [26]	A centralized congestion policy to mitigate Incast.

storage servers are increased to 40. This resulted into buffer overflow, which caused retransmission and latency. In addition, the latency degrades performance and throughput. Phanishayee *et al.* [5] measured and analyzed TCP throughput collapse (storage protocol uses TCP for transport) in a cluster based storage system, and proposed their solution at the Ethernet level—enabling Ethernet Flow Control to prevent Incast. They further reported that TCP slow start failed tackling Incast. Another attempt is done by Chen *et al.* [16], where authors proposed limited number of senders to handle Incast. Although, it solves the problem. However, this approach fails scalability and practical use. In a data center storage environment, it is difficult to predict the number of active senders for a given time.

Zhang *et al.* [27], [28] have tackled Incast through addressing the block tail time out (BTTO, sender cannot send further until all senders finished sending the data) and block head time out (BHTO, when many senders send and some finishes sending soon). They modified TCP behavior and prevented these two timeouts (they claimed that these timeouts are the real cause of Incast). A comprehensive overview, to understand factors contributing to Incast, can be found at [29]. Performance evaluation of TCP congestion control can be found at [30]. A list of mostly used Incast methods and their brief description is presented in Table 1.

Alizadeh *et al.* [17], and Wu *et al.* [18] proposed DCTCP, and ICTCP. These are two early window-based approaches preventing Incast. DCTCP was designed to work in a data center; it requires changes both to ECN enabled switch and TCP at the sender nodes [31], [32]. DCTCP does not consider application level priority. DCTCP uses ECN signal for congestion notification—a simple mark at the ECN switch, and an ECN echo at the receiver. Buffers allocation are controlled from a sender node. DCTCP is limited in scalability as it surrounds only 35 concurrent senders. In contrast to DCTCP, ICTCP uses the receiver buffer size for controlling buffer allocation. Both these methods use buffer policy at the sending receiving node —host based solution. They lack a centralized controller approach. Zhang *et al.* [33] proposed Adaptive Marking Threshold (AMT) to proactively tune the marking threshold for eliminating the queue delay in the data center environment. They mentioned that the fixed ECN marking fails to handle the queuing delay. This is a switch based solution which lacks the centralized approach. Managing and tunning individual switches in a large data center introduces administration and maintenance overhead. Reducing Time Out (TO) can also mitigate Incast. Vasudevan *et al.* [19] proposed RTO based solution; they decreased the default TCP time out, however, this technique leads to a fast retransmission which can saturate the network quickly. A very low RTO increases premature times-outs [5], and, thus incurs quick buffer exhaustion and Incast [1].

Zhang *et al.* [20] have shrunk the MTU value to mitigate Incast. However, reducing MTU for long flows suffer performance and link degradation. *RTT* based congestion control mechanism is proposed by Mittal *et al.* [21]. They claim *RTT* is a holistic signal compared to ECN.

Receiver-oriented Congestion Control (RCC) mechanism is proposed by Xu *et al.* [34]. However, RCC is a receiver-host based congestion control mechanism. This scheme did not consider the advantages of SDN approach. Another work is done by Huang *et al.* [35] where authors adjusted the packet size to mitigate Incast. A recent work done by Almasi *et al.* [36]. They proposed Explicit Incast Notification (EIN) opposed to ECN. Authors claimed that as compared to ECN, which is based on slow start, EIN is fast and accurate. Their scheme is called Pulser, and the EIN is governed by switch. The drawback of this approach is the decentralized control mechanism which is incorporated by all connected switches in the data center. Remote Direct Memory Access approach is proposed by Xue *et al.* [37], where authors explained that the approach drastically decreases the latency. In this approach, TCP packets need not to traverse the whole operating system stack.

B. INCAST IN CONTEMPORARY STORAGE SYSTEMS

Ghobadi *et al.* [25] presented openTCP, a software-defined inspired way that augments DCTCP and other protocols to prevent congestion. openTCP surrounds a general network, while DyFaShBuP surrounds a storage network. The congestion implementation policy in DyFaShBuP is another difference with openTCP. In our scheme, the congestion control mechanism and dynamic fair-share buffer allocation for end-to-end I/O path is dependent on the number of concurrent connected servers, priority queues, *WS*, and bandwidth delay product (*BDP*). An arbitrator mechanism handling Incast is proposed by Munir *et al.* [13]. They used arbitrator at each stage for feedback. Multiple arbitrators result arbitration delay, and processing overhead.

Another SDN implemented method, SDTCP, is reported by Lu and Zhu [38]; they proposed a technique where

TABLE 2. Notations used in the proposed method.

Notation	Description
BW	Bandwidth, link capacity.
WS	Window size, a performance parameter used to specify data blocks that are still un-acknowledged.
FS	File stripe, the size of a single stripe is measured in bits; each file consists of multiple stripes.
BDP	Bandwidth delay product, the product of available BW and latency yields in the amount transit data in the network. BDP = link speed in bytes $*$ RTT in seconds.
RTT	Round trip time, the time required for a complete round from sender to receiver and back.
CFS	Concurrent file servers, a performance metric denotes the number of concurrent senders.
PF	Prioritized flow, a data flow given a high priority than a normal flow.

a congestion signal is generated by a switch and forwarded to the controller. DyFaShBuP handles the storage side, while SDTCP worked on the network side. In SDTCP, the queue size is checked all the times for buffer availability in all switches. This results into a scalability issue. SDTCP handles Incast for 50 senders. A switch with ECN mechanism tackling Incast is proposed by Lee *et al.* [23]. In their method, a switch is an intelligent device and responsible to prevent Incast. In a data center environment, managing multiple switches through this scheme lacks manageability and proper maintenance. As more switches are added, more installation and upgradation are needed. Another SDN data center switch based method is proposed by Hwang *et al.* [24]; they provided Incast solution via SDN switch and not via SDN controller.

SDN controller based (OTCP) Incast solution is proposed by Jouet *et al.* [26]. This approach surrounds data center network's aspect. Their proposed topology poses a significant processing and traffic overhead on the controller. Qin *et al.* [39] presented TCP with Acknowledgment Changing Rate (TCP-ACR) to mitigate Incast. Their congestion control method is based on the changing rate of ACK, and the experiment is only limited to 39 servers (a potential scalability issue). Wang *et al.* [40] presented a solution (TCP-FIT) for a heterogeneous network (both larger BDP and wireless links).

All these different techniques provide different solution sets depend on the application environment. However, none discuss the Incast issue for storage network. The key notations are summarized in Table 2.

IV. ANATOMY OF INCAST

We understand that Incast is not related to a particular topology setup. It is an issue where multiple concurrent senders are bombarding a single receiver whether it is a storage device, network device, or any other communication entity. Incast is dependent on BW, latency, buffer capacity, I/O stack, RAM, and CPU. It surrounds the whole system. In this section, we dissect Incast in details.

- Incast and buffer: Bechtolsheim *et al.* [41] proposed and claimed that big data needs a big buffer, and as a result, it solves the Incast. In contrast to [41], Alizadeh *et al.* [17] claimed that big buffers provide a small opportunity. A shallow buffer switch cannot handle Incast, and thus, degrades the performance (they build queues for long flows). As the buffers get deeper, it leads to a higher latency. In addition, switches with larger buffer are costly [5].
- Incast and switch port: Incast is also dependent on the deployed switch architecture (port). Modern switches have dedicated ports (port-based memory buffering) and shared buffer ports (shared memory buffering). There is a trade-off between these two. Shared buffers facilitate better burst absorption, because a large dynamic buffer pool can handle the burst as needed. However, a larger shared buffer leaves a small capacity for dedicate ports. A dynamic fair-share buffer policy is needed tackling Incast. In contrast to shared buffer switch architecture, a dedicated buffer guarantees equal space to each port, and thus, provides a fair-share buffer. However, small dedicated buffers lead to Incast—they cannot handle a traffic burst for a short time. Storage systems contain both long flow and short flow. The underlined switch architecture should address these issues.
- Incast and line rate latency: If the deployed link latency is high, storage system will face Incast. For 50*ms* line-rate latency, a switch with 10Gbps needs 60MB of buffer. Smaller the line-rate latency, higher will be the throughput, and vice versa. Defining the exact amount of buffer capacity is difficult. There are numerous switch design issues (dedicated, shared, and hybrid) that make it hard. Incast latency factors include: distance, the deployed equipment's performance, the congestion algorithm, and the protocol.
- Incast and blocking and non-blocking switch architecture: Device buffer capacity can be limited by hardware architectures. A blocking architecture fails to meet the full-duplex BW requirements. This case leads to a packet loss. In contrast to blocking architecture, a non-blocking architecture provides an extra internal buffer capacity to meet the full-duplex BW requirements of its port. It guarantees no packet loss; and it does not affect other links in the deployed switch.
- Incast and high resolution timer: Due to a higher resolution timer in many OSs, reducing RTO from 200*ms* is not possible [5]. Reducing RTO helps TCP to enter in the retransmission mode quickly to deal with the congestion. However, RTO should not be decreased too much. It increases the retransmissions drastically and ultimately chokes the BW.
- Incast and BW: Incast is directly related to the available BW. A smaller BW leads to Incast, while a sufficient BW prevents it. However, predicting the optimal BW and latency requirements for tenants are still unsolved problems [42].

- Incast, and long flow and short flow: A tremendous amount of data flows in storage systems. These consist of long and short flows. Short flow remains for a very short time, while long flow takes more time. The stripping behavior in a storage system can affect these flows. *FS* and data block size should be practiced and experimented for a better result.
- Wider stripping: The wider the stripping, the maximum is the performance [1]. However, a wider stripping needs higher number of storage servers. Wider stripping provides concurrent simultaneous data access. Without dynamic fair-share buffer allocation, it leads to Incast. Section V and VI elaborate more on wider stripping.
- Incast and *BDP*: *BDP* determines the amount of data that a link occupies at a given time. It is directly related to latency and throughput. If the *BDP* is less than the TCP *WS*, the path BW is the limiting factor for maximum throughput. In another case, if the TCP *WS* is the limiting factor for throughput, we use multiple concurrent TCP connections to fill the pipe. In a storage system, to prevent Incast, we should use a proper relation between *BDP* and the underlined protocol *WS*. When the network topology and routing change over time, we should periodically check the *WS* for maximum performance.
- Incast and *WS*: One of the most important performance factor in our proposed algorithm is the use of *WS*. It represents how much data can be sent from one storage host to another without being acknowledged. It is the amount of data on a link at a given time. A *BDP* is the theoretical value for the TCP *WS*. *WS* cannot be greater than *BDP*. Because of the OS and network implementation, tuning an optimal TCP *WS* is a tedious job. Some misbehaved *WS* may degrade storage performance.
- Incast and memory: Every TCP connection requires a memory space. An increased *WS* demands more memory on the server. All un-acknowledged packets should be held in memory, which result in more memory demands. A faster main memory provides a faster I/O access. Islam *et al.* [43] have introduced NVM based burst buffer technique to improve I/O performance. In their design, the communication buffer acts as a storage buffer, which consequently increases write performance. Although, they have not discussed storage Incast, but their proposed method supports mitigating Incast in an extensive I/O storage cluster. For a small scale storage cluster, burst buffer may not be needed. However, the burst buffer will be a mandatory component for peak I/O rates in a massive storage cluster [44].
- Incast and multiple TCP connections: Multiple TCP connections provide the advantage of filling the pipe. Lowering the *WS* than the available *BDP* under-utilizes the link capacity. Using millions of TCP connections require a considerable amount of memory. However, only memory will not help; a device must support greater connections per second to get the full advantage of memory.
- Incast, TCP, and storage protocol: To prevent Incast, we should address all the entities involving in the communication (e.g., iSCSI). The underlined storage must be fast enough to handle the maximum IOPS. Any mismatch results into Incast. Storage devices should not be the bottleneck for the available link *BW* and vice versa.
- Incast and queue: This is related to switch architecture. Multiple queues provide more space and better flow control; however, it incurs more cost.
- Incast and concurrent senders: Incast is directly proportional to the number of concurrent senders. Chen *et al.* [16] limited the number of concurrent sender to mitigate Incast. We should find a better solution for concurrent senders to help the scalability factor of storage system. In addition, the amount of traffic, size of packets, and network diameter are also the affecting factors towards Incast.
- Incast and file system: As stated that to prevent Incast, we should consider the whole system, and not only storage system or networking part. The underlining file system (parallel and distributed) also contributes to Incast. Poorly tuned multiple parallel I/O requests from multiple servers can degrade performance [45], and lead to Incast. Figure 2 shows it clearly. Although big data-blocks and stripe size provides maximum throughput. However, they increase pressure on the available buffer capacity. In addition, the overall performance is limited by I/O stack, disk latency, and stripping and spreading mechanism.
- Incast and disk technology: I/O performance is still far behind the computing power in modern data centers [3]. The gap between I/O speed and computing power decreases the overall I/O throughput. It is suggested to use state of the art high access disk technology to prevent Incast. Incast situation even becomes worse when multiple I/O accesses are requesting multiple I/O devices.
- Incast and NIC: The installed NIC must support a high throughput to prevent Incast—we should not introduce another bottleneck.

  An abstract Figure 4 shows different stages that contribute to Incast. Providing a dynamic fair-share buffer for end-to-end I/O path and maximum I/O throughput should understand the complex connectivity of modern data centers. A high performance I/O path demands a high performance system—both network and storage must meet end-to-end I/O requirements.

V. ARCHITECTURE AND PROPOSED MODEL INSPIRED BY SD CONCEPT

TCP Incast is extensively discussed in the literature. Our main focus is the integration of software-defined concept in a data center storage cluster while tackling Incast.

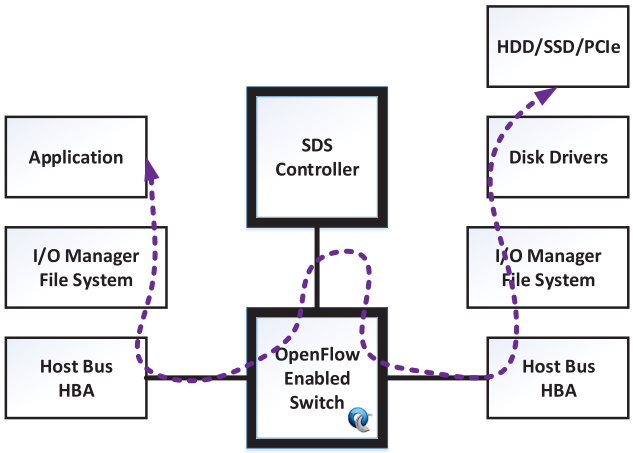

FIGURE 4. Tackling Incast in end-to-end I/O path: Multiple factors contribute to Incast, while handling an end-to-end I/O path. Guaranteeing an end-to-end I/O path should consider all stages.

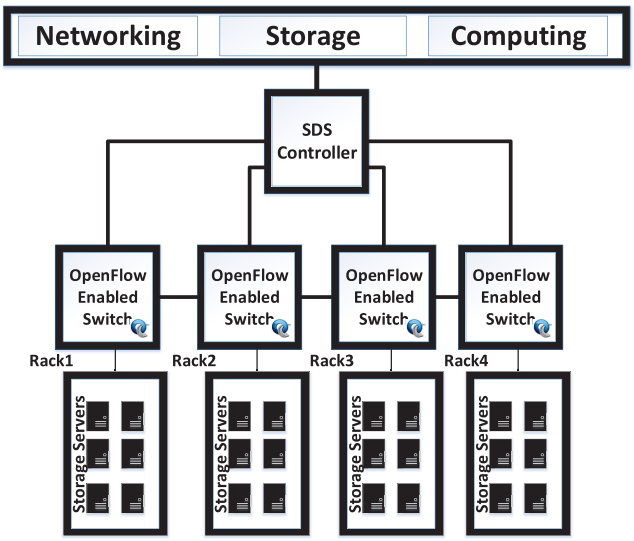

FIGURE 5. The abstract topology: A SDDC consists of Computing, Storage, and Networking. The storage topology consists of SDS controller, OpenFlow enabled switches, and 48 storage servers per rack.

We provide a protocol that can be scaled, configured, and troubleshooted easily. The common topology is depicted in Figure 5. In this figure, every switch collects statistics of the designated rack's servers and reports to the storage controller for policy enforcements. The link BW between storage controller and OpenFlow enabled switch is 10/40G, while the link between storage server and OpenFlow enabled switch is 10G/1G. We incorporated Top of Rack (ToR) design (not a mandatory policy)

In DyFaShBuP, we model the most common packets in OpenFlow protocol specification: Packet-In is a packet that requests the SDS controller to handle a flow. Packet-Out is the response from SDS controller to switch to send that particular flow. Packet-Mod tells the switch to install a new flow table entry. The Packet-Exp instructs the flow time-out after a certain period of inactivity.

A. SDS CONTROLLER

SDS controller is the core entity in DyFaShBuP. It is a high quality hardware that provides software-defined storage services. It has 10GE/40GE/100GE ports according to the storage demands. It has a secure communication channel to interact with switches. Communication channel is protected to ensure cryptographic components, e.g., confidentiality, integrity, and availability. SDS controller also incorporates meta data services and a global cache service. However, these two concepts are out of scope of this work (our future target). The advantage of global cache server is the success hit. In such a case, data stripes will not be fetched from individual storage nodes. Preventing Incast in a storage network, the SDS controller responsibilities are as follows.

- It dynamically calculates the number of attached storage nodes (based on IP or MAC address).
- It calculates the file system's block-size and the FS (file data to be stripped on), where FS=Block-size/no of attached storage servers. For high performance, a wider stripping is needed. Wider stripping provides multiple concurrent I/O accesses simultaneously.
- Because the DyFaShBuP provides a global flow awareness, it facilitates storage behavior observations and, thus, helps defining cold nodes and hot nodes. Cold nodes have less chances to be accessed while hot nodes have maximum I/O access chances.
- Observing the overall storage and network behavior, SDS controller defines a DyFaShBuP. The policy includes: global flow awareness, dynamic fair-share buffer allocation for end-to-end I/O path, and global fair-share BW utilization. This policy is distributed to storage servers via OpenFlow enabled switch. Subsection V-D is dedicated to discuss the most important contribution of our method—the dynamic fair-share buffer allocation for end-to-end I/O path.
- SDS controller adjusts the policy according to the underlined switch supported architecture (shared buffer, dedicated buffer, and the hybrid buffer).
- On the application request, it defines the buffer for long flow, short flow, and PF.
- The extension of OpenFlow protocol provides to accommodate multiple TCP connections, and statistics. Upon receiving these statistics, the SDS controller updates the advertised window in the TCP header.

B. OPENFLOW ENABLED SWITCH

A high processing power enabled switch is recommended for maximum I/O throughput in a data center storage cluster, which can facilitate multiple concurrent I/O requests from multiple servers. A hardware system should not be another bottleneck when tackling Incast. In our proposed method, the switch responsibilities are as under:

- DyFaShBuP uses both normal and priority queues. The priority queue is used to prioritize a flow. The SDS controller implements the policy, while the switch executes it.

- We have defined a policy table for handling storage flows. This policy table consists of match-action entries. When a miss action-match occurs, switch requests the SDS controller to handle the new flow. SDS controller defines a new policy and forwards to the switch. Further research is needed whether to have multiple software enabled policy tables or hardware policy tables. The flow chart for policy table implementation is shown in Figure 6.
- Our proposed switch consists of 48 ports. Each port supports 1GE. The switch architecture is out of scope of this work. However, we anticipate a shared buffer and a dedicated buffer switch for the proposed scheme.
- Each TCP session reserves a physical memory matching the buffer size. Because of the switch memory limitation, we do not use unlimited TCP connections.
- In DyFaShBuP, triggering a congestion signal is not the responsibility of a switch as in [38], but we incorporate the central controller to tackle the congestions. Another important point is the target flow identification. We delegate this policy from switch to controller. A centralized controller-based target flow identification eases the management and maintenance. Switch based methods lack this advantage. It leads to multiple switch configuration, troubleshooting, and maintenance.
- A dedicated port buffer can prevent head of line blocking [27]. However, we use a dynamic buffer policy to tackle Incast. We are not aiming to change the switch architecture.

C. STORAGE SERVER

A storage server has all the traditional properties, e.g., storage capacity, storage technology (HDD, SSD), and number of supported NICs. NIC should not be the bottleneck. If the link BW is 1Gbps, and the installed NIC supports 500Mbps or less than 1Gbps, definitely, this bottleneck will result into Incast.

D. GLOBAL BUFFER DYNAMICS

An ideal congestion control scheme for DCNs should efficiently use the network resources, e.g., BW, and buffer capacity [30]. DyFaShBuP for end-to-end I/O path offers a unique way to tackle Incast. It calculates the complete I/O route from end-to-end system. All the communication entities are taken into consideration while applying the global calculation formula. It is a policy that prevents Incast whether the root cause is TCP protocol itself, switch, or the end storage server. This calculation results into a dynamic fair-share buffer allocation, maximum overall I/O throughput, and a low latency.

A single end-to-end I/O request traverses multiple I/O stages as shown in Figure 4. Enforcing policies at each layer are hard for I/O [12], [46]. They have shown a general I/O flow architecture, while we aim to prevent Incast via a dynamic fair-share buffer allocation for end-to-end I/O path. DyFaShBuP adjusts the WS across all I/O paths—from all sending file servers to the receiving file server. Different stages are shown in Figure 4. These expose their statistics to the switch, which is then forwarded to the SDS controller. In addition, the switch collects other statistics to enforce fair-share buffer for end-to-end I/O path: number of concurrent senders, file server receiving and sending buffer capacity, switch buffer capacity, link BW and latency, BDP, and the supported disk read and write operations' speed. SDS, then defines the global buffer policy as shown in Figure 6. Controller instructs all the CFS to adjust their WS as directed—**Buffer dynamics is the control policy of SDS controller. This policy is not based on the receiver, sender, or switch (the new approach).** DyFaShBuP is completely software-based. We are not proposing any hardware change. We are decoupling the TCP default window mechanism and delegating it to the SDS controller. Different control policies are shown in Table 3.

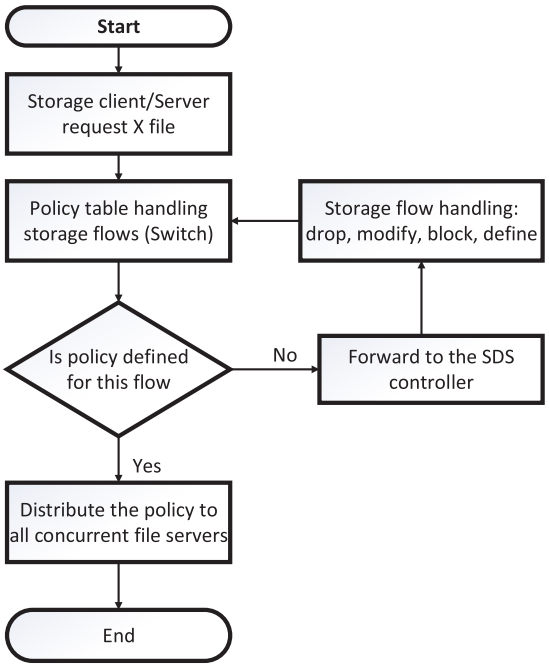

FIGURE 6. A simple flow chart demonstrating the global policy implementation for storage flows.

Our method eliminates the specific time granularity, e.g., some time interval granularity time (gt) to update the policy via OpenFlow enabled switch. Whenever a system's parameter changes, it is directly propagated to the SDS controller, and the SDS controller implements the policy. A common granularity parameter gt can be defined to update the global policy. However, gt incurs extra delay. Our dynamic policy does not rely on gt. Compared to the send/receive window based method, e.g., DCTCP and ICTCP that can adjust the window size once detecting the congestion, our scheme implements the policy prior to congestion detection. Through our dynamic policy, the SDS controllers eliminate the Incast mitigation delay—there is no gt interval to update policy.

TABLE 3. A dynamic fair-share buffer for end-to-end I/O path allocation policy: Different functions are used to implement different policies.

Policy	Description
Dynamic fair-share buffer calculation	A global fair-share buffer for end-to-end I/O path calculation is based on the sender's buffer size. It includes: disk I/O, client memory, NIC parameters, link BW, packet size, BDP, RTT, and switch buffer capacity.
Dynamic fair-share buffer allocation	Based on the collected statistics, controller distributes the WS to all CFS. This policy provides a dynamic fair-share buffer and BW for end-to-end I/O path.
Priority based traffic	On behalf of application priority, controller distributes the policy to all CFS.
Rate limit buffer allocation	For administrative and control purposes, controller limits the buffer allocation based on end-to-end buffer calculation, and distributes the policy to all CFS.

DCTCP and ICTCP use end-system buffer policy. Either the sender or the receiver governs the policy (*WS* adjustments). The drawbacks of these two approaches are already discussed. In addition, our approach is a decoupled-inspired approach to mitigate Incast, while DCTCP and ICTCP are host based approaches, and completely missing the decoupled-architectural advantages. DyFaShBuP uses a centralized mechanism defining a dynamic fair-share buffer for end-to-end I/O path. A buffer congestion mechanism used by either end systems (DCTCP, ICTCP), or switched based system, leads to a tremendous amount of communication overhead (decentralized buffer coordination by end hosts). We eliminated the extra communication through a centralized SDS controller. Our scheme achieves the maximum overall I/O throughput.

1) BANDWIDTH DELAY PRODUCT (BDP) CALCULATION

For the sake of understanding, we elaborate the bandwidth delay product calculation for the fair share buffer management as under, where, $Bandwidth = BW$, and $delay = Delay$. BW is measured in bits per second, and $Delay$ is measured in seconds. $BDP = BW * Delay$ If we have $1Gbps$ link, and $1ms$ Delay for that link, then the BDP is as follows: $BDP = 1*10^9*10^{-3} = 10^6 bits = 125KB$ This means, at one particular time interval, i.e., for $1ms$, the maximum bytes that can be in transit on this link are $125KB$. To fully utilize the $1Gbps$ link, the maximum WS must be $125KB$. As this is the maximum achievable BW, we cannot increase this size more than 125 KB of (WS which is alway bounded by the BDP). We can lower this value from $125KB$, however, in that case, we will be under utilizing the link bandwidth, which is not an optimal choice. For hight throughput, we always use the maximum supported WS in order to fully utilize the link bandwidth. A graphical representation is given in Figure 8.

FIGURE 7. The software agent is responsible to collect different information (IP, Buffer capacity, etc.), and share with SDS controller.

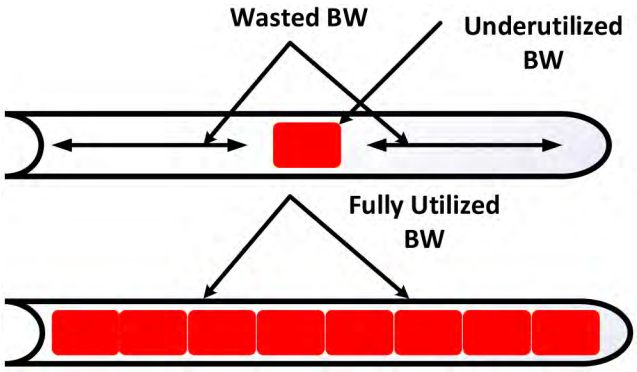

FIGURE 8. A small window size incurs bandwidth underutilization. Increasing the window size guarantees the maximum available throughput.

2) ALGORITHM OVERVIEW

The Algorithm 1 shows how dynamic fair share policy works for all the concurrent storage servers. Initially, the program accepts the number of storage servers as integers, delay in *ms*, and bandwidth in *bits per second*. The method will calculate the bandwidth delay product and this whole product will be divided by the number of concurrent file servers, which will yield the window size WS. The SDS controller will distribute this WS to all connected servers.

3) IMPLEMENTATION FEASIBILITY

As the proposed solution is wholly based on software via a dynamic fair share policy, which is governed by the centralized SDS controller, its implementation is feasible from a small to a big data center. There is no need to modify the client TCP legacy implementation. A small size buffer share policy agent can be installed on those servers which will communicate with the centralized SDS controller for the Window Size behavior and negotiation. The architecture is shown in Figure 7. The agent will listen to the SDS controller, and will fine tune the Windows size for that particular server. DCTCP only applicable if the switch supports ECN.

Algorithm 1 Fair Share Buffer Allocation in a Datacenter Storage Cluster

Input: available BW in bps, $delay$ in $milisecond$, number of attached storage servers
Output: BDP in bits or Kilo Bytes, window size WS in KB

/* Number of attached storage servers $n = 48$, user can input any number for n from 1 to 48. Number of attached open flow enabled switches $ns = user - input$. We assume user inputs 20 for this variable, so the number of available switches are 20. We have 20 switches, and 48 servers attached to each switch. Note that both n and ns can have run time values*/
$n = 48, ns = 20, BW = x, delay = d, BDP = 0, WS = 0$;
for $k = 1; k \leq ns; k++$ **do**
  for $i = 1; i \leq n; i++$ **do**
    float $BDP = x * d$ /* bits or Kilo Bytes*/;
    float $WS = BDP/n$;
    distribute the WS to attached servers i connected to switch k;
  end
end

4) FAIR-SHARE BUFFER MANAGEMENT

It is crucial to have a sufficient buffer capacity to handle micro-burst caused by many-to-one communication pattern. The default TCP flow drops a packet due to the buffer overflow. We aim to maintain the smooth operation and prevent TCP from sticking in the recovery mode. We have a 48 port switch, and 48 file servers are connected through 1GE. Each file server has 4 GB RAM. Switch supports shared as well as dedicated buffers. We assume a 200 KB capacity per port. We incorporate a latency of $1ms$ for data storage network. DCTCP uses $250us$ latency for data center for a non-congested network.

With such parameters, BDP is the limiting factor (WS is always bounded by BDP). If the BDP is less than switch's dedicated per-port buffer capacity, then BDP is the bottleneck. We cannot use the full 200 KB port capacity with a less BDP. For the maximum I/O throughput, BDP should equal to buffer capacity. In the same way, if the underlined protocol does not support a sufficient amount of WS, then the WS can be the bottleneck, no matter how big is the link. We have tackled both cases. For the link $BW = 1$Gbps, and $RTT = 1$ms, the $BDP = 125$ KB. With the default TCP windowing mechanism, the receiver notifies how much data should be sent in the next window, or what will be the next WS. DyFaShBuP uses the centralized controller for calculations.

If 20 file servers are sending at one time to only one server, the DyFaShBuP for end-to-end I/O path divides the maximum buffer capacity by the number of outstanding CFS. In this case, it yields 125KB/20 = 6.25 KB. This value suggests that all the CFS should use 6.25KB of window, not more than that—otherwise the TCP Incast will happen. 125KB is the maximum buffer capacity that a receiver can support at one-time instance. It is bounded by BDP. As 125KB is supported by all links (because of BDP), 20 file server cannot fully utilize the whole BW. However, we achieve the maximum overall I/O throughput. In fact, the aim is the overall I/O throughput and preventing Incast. BDP works only for a single TCP connection. We aggregate the value for multiple connections. In the above example, to fill the whole pipe with 125KB, all 20 senders should send at one time with 6.25KB WS, and this proves our calculation. Multiple factors govern the I/O throughput: CFS cannot send more data at one time than the SDS controller advertised buffer capacity; CFS cannot send more data than their supported WS; CFS cannot send more data than the BDP; and CFS cannot send more data than the available data in the sender buffer.

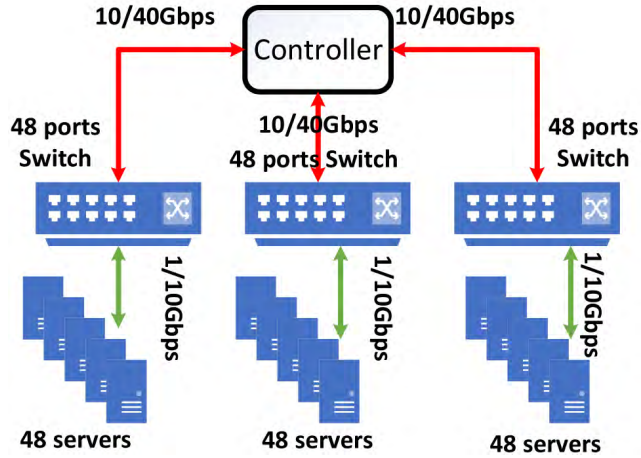

FIGURE 9. The general setup shows a centralized controller with 10/40Gbps connected to switches, which are further connected to 48 servers via 1/10Gbps.

VI. PERFORMANCE ANALYSIS AND EXPERIMENTAL RESULTS

In DyFaShBuP, the global buffer policy implementation provides both flow control and a fair-share BW utilization. The implementation setup is shown in Figure 9.

A. INCAST SCENARIOS

THROUGHPUT VS CONCURRENT FILE SERVERS

Incast can be caused by different parameters as explained in Section IV. In our proposed work, we are focusing on the effect of concurrent file servers over throughput. That is the case we only showed these experiments. Experiments over other parameters are beyond the scope of this work; working on those parameters are our future tasks.

In a production storage level, Incast spans over multiple switches and racks. We categorize Incast into three scenarios.

- Within a rack (single switch): File server1 in rack1 requests FSs from multiple file servers within the same rack, and through the single switch.

TABLE 4. Simulation parameters in DyFaShBuP.

Parameter	Value
Server-switch BW	1/10 Gbps
Controller-switch BW	10/40 Gbps
RTT	$1ms$, $1.1ms$, $1.2ms$
Number of CFS	1-48 (not limited)
WS	125/137.5/150KB
Number of ports	48
Number of SDS controller	1
Number of switches	Multiple
Switch to switch BW	10Gbps
Switch buffer	200KB

- Single rack to single rack (two switches): File server in rack1 receives *FS*s from all servers in rack2. In a real world, file server1 in rack1 may receive from multiple servers from rack1 and rack2 simultaneously.
- Single rack to multiple racks (more than two switches): File server1 in rack1 can request *FS*s from multiple file servers in different racks through multiple switches.

B. EXPERIMENTAL RESULTS

Different simulation parameters are shown in Table 4. NICs in all file servers are fast enough to accommodate the overall link throughput. All experiments were performed in Matlab programming environment. We have run the program 100 times to validate our results. Results presented have been averaged out over the course of simulation. As far as the confidence is concerned, we run the program on Dell e6440 Latitude, 8GB RAM, 256 GB Samsung SSD, and 1TB western digital hard disk. In all these provided parameters and our system specification, the reproducible results will be same on other machine. However, a small variation in reproducible result may occurs due to system's specification and other parameters.

- Single file server requests *FS*s from multiple file servers (same rack): We have done multiple experiments to calculate the overall end-to-end I/O throughput when a single switch governs the communication among file servers. Figure 10 shows the experimental results. In this case, the minimum *WS* is 2.6*KB* for all 48 *CFS*. Maximum *WS* is 125*KB* for a single file server. The Maximum I/O per link with 125*KB WS* is 1*Gbps*. Minimum I/O per link when all file servers are sending concurrently is 21.30*Mbps*. We achieved the overall I/O throughput near to the available link capacity—no matter how many *CFS* are sending. Following values are incorporated according to the available network capacity: $RTT = 1ms$, $BW = 1$Gbps. To fully utilize the receiver buffer (125KB), the SDS controller uses 125KB *WS*. This *WS* should be equal or greater than *BDP* for maximum I/O throughput. SDS controller allocates a dynamic fair-share *BW*, and *WS* for end-to-end I/O path preventing Incast, while at the same time, achieves a maximum overall I/O throughput.

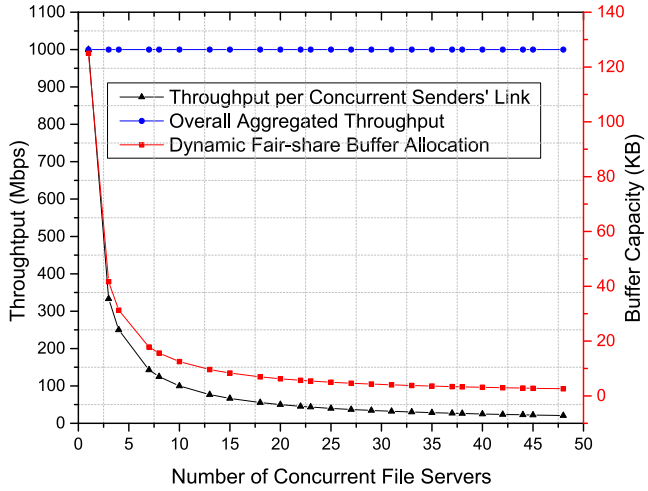

FIGURE 10. A Single file server requests *FS*s from multiple file servers.

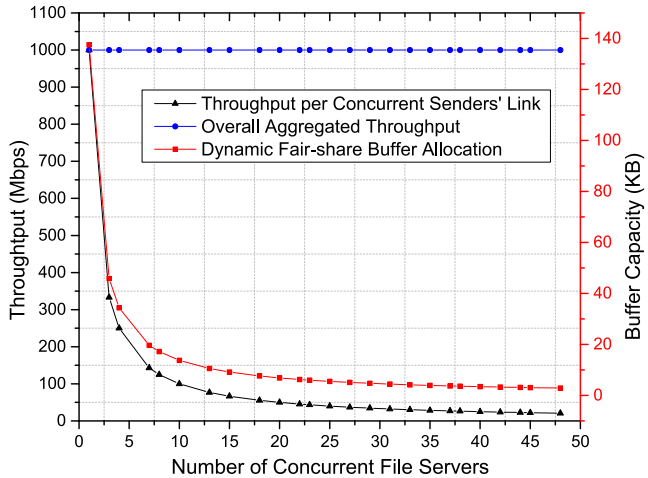

FIGURE 11. A single file server requests *FS*s from multiple file servers through a second switch.

- Single file server requests from multiple file servers (A different rack, two switches): The single switch communication for storage is the simplest one for Incast understanding. A more realistic scenario can be seen in Figure 11, where two switches are incorporated. Minimum *WS* is 2.86*KB* for all 48 *CFS*. Maximum *WS* is 137.50*KB* for a single file server. The Maximum I/O per link with 137.50*KB WS* is 1*Gbps*. Minimum I/O per link when all file servers are sending concurrently is 23.43*Mbps*. In this case again, via the SDS global buffer policy, we achieved the overall I/O throughput near to the available link capacity—no matter how many *CFS* are sending. For this case, we incorporated the following values according to the available network capacity. $RTT = 1.1ms$, *BW* for file servers link = 1Gbps, *BW* for switch to switch = 10Gbps. For the end-to-end I/O throughput, SDS controller considers all communicating entities, i.e., file server buffer capacity, switch buffer capacity, *RTT*, and BW. For 1.1*ms* latency,

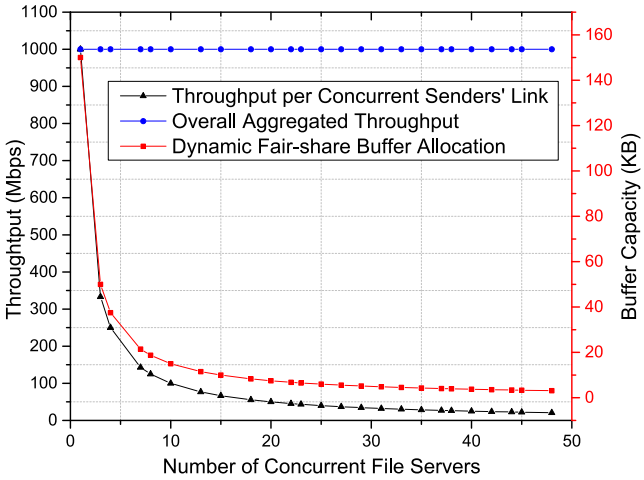

FIGURE 12. A single file server requests *FS*s from multiple file servers through multiple switches.

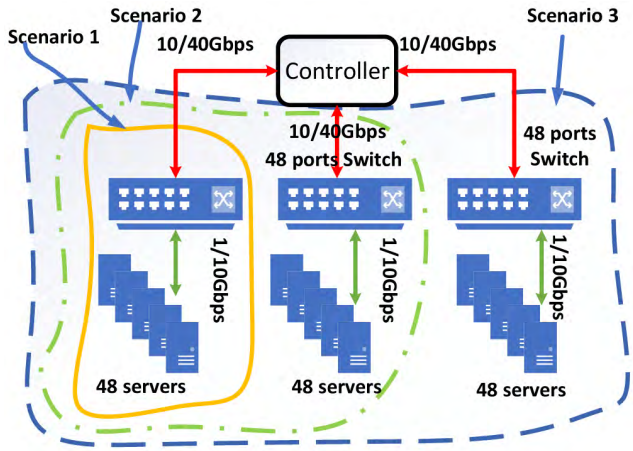

FIGURE 13. Scenario 1 shows single file server requesting from multiple file servers within the same rack. Scenario 2 and Scenario 3 are self explanatory.

the *BDP* is 137.5 KB. For maximum I/O throughput, file server supports 137.5 KB buffer (it can support high buffer as well). The deployed switches have an enough buffer capacity to accommodate the *BDP*. In this case, the SDS controller assigns 137.5 KB of *WS*, and distributes among all *CFS* to prevent Incast as shown in Figure 11. In this case again, via the SDS global buffer policy, we achieved the overall I/O throughput near to the available link capacity—no matter how many *CFS* are sending

- Single file server requests *FS*s from multiple file servers (different racks and multiple switches): We have done experiments for the worst case of Incast where a single file server requests from multiple file servers through multiple switches and racks. The result of DyFaShBuP for end-to-end I/O path can be seen in the Figure 12. Minimum *WS* is $3.125KB$ for all 48 *CFS*; Maximum *WS* is $150KB$ for a single file server. Maximum I/O per link with $150KB$ *WS* is $1Gbps$; Minimum I/O per link when all file servers are sending concurrently is $25.60Mbps$. Following values are incorporated according to the available network capacity. $RTT = 1.2ms$, *BW* for file servers link = 1Gbps, *BW* for switch to switch = 10Gbps. File servers support the buffer capacity bounded by *BDP*(150KB); the *BDP* is 150KB; and the *WS* is as much as *BDP* to avoid Incast and link under-utilization. SDS controller distributes the *WS* over multiple *CFS* for fair-share BW and buffer allocation. The DyFaShBuP prevents Incast as shown in Figure 12. Different scenarios are shown in Figure 13.

C. PERFORMANCE ANALYSIS

As stated earlier, *BDP* governs the amount of data that can be sent in the pipe at a given time. A sufficient *BDP* provides an optimal value for the *WS*. Increasing TCP *WS* or reducing latency provides higher link utilization. However, without proper tuning, this approach leads to Incast. In DyFaShBuP, the automated *WS* tuning by the SDS controller facilitates the optimal buffer capacity for different Incast scenarios as illustrated in the Section VI-A. The global buffer policy allocation always provides an end-to-end fair-share I/O *BW* and a fair-share buffer preventing Incast in the data center storage cluster—it considers the whole I/O path.

However, due to the complex data access pattern (small vs big) in a data center storage cluster, data-blocks span over multiple storage nodes can degrade I/O performance. A small I/O choice may be suited for one situation, while a big I/O will be better for another [47]. We observed that, although, a small *WS* for 48 file servers achieves the line-rate I/O throughout—not a goodput, however, it leads to link-underutilization. To prevent this, we suggest a small file (some threshold) should not be stripped over many storage servers.

There should be a policy in a file system, e.g., that a file less than X MB should not be stripped over N servers. To understand data-layout (small vs big), it is important to use an optimal MTU—the smallest amount of data that a protocol can send in one packet—in the storage network. The maximum MTU supported by an Ethernet interface, excluding Ethernet frame header and trailer, is 1500 Bytes. It includes 1460 Bytes for MSS, and 20 Bytes each for IP and TCP header—a 40 Bytes overhead in each packet. Therefore, a very small *WS* for a small file or a large file degrades the I/O throughput. Sending a very small data size results into a very large overhead, and sending a very large data results into fragmentation. Figure 14 shows the impact of *WS* and its traffic overhead. However, a small *WS* introduces link under-utilization.

Our policy of fair share buffer dictates that whether we use per port buffer architecture or shared buffer architecture, the fair share policy will allocate an equal and fair amount of bandwidth to all concurrent connections. It will not be the case that one server or one connection will be utilizing

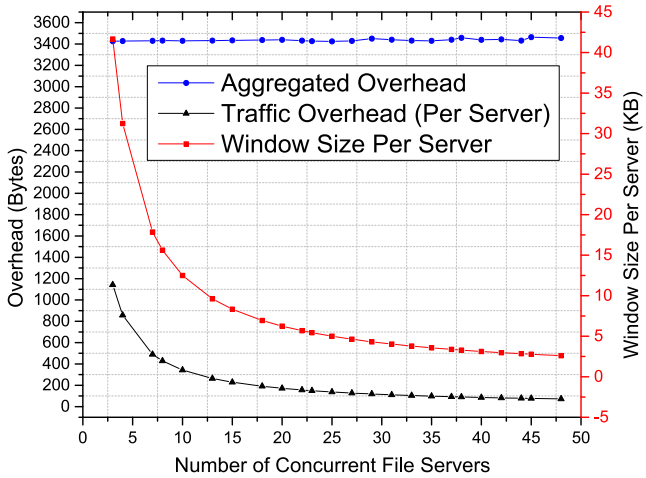

FIGURE 14. Impact of *WS* and overhead: The aggregated overhead is almost constant, 3400-3460Bytes or 2.75 % whether we use a small *WS* or a big *WS*.

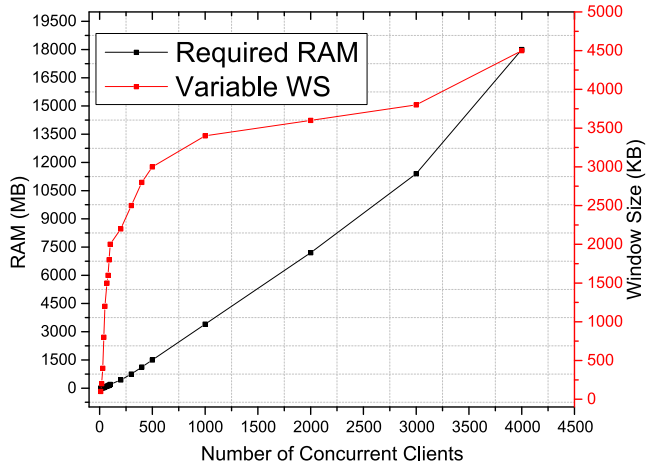

FIGURE 15. Impact of *WS* and concurrent requesting clients: For a constant *WS*, increasing number of concurrent clients demand more RAM equals to *WS* times the number of CFS.

70% of the link bandwidth, and the other connection will be using merely 30% of the link bandwidth. Even in the worst case, if we have 100 servers, the bandwidth share for all these 100 server will be fair, i.e., equal for all. In case of 100 servers, if one server is utilizing 5Mbps, all remaining servers will also utilize 5Mbps. Our proposed method will work whether we use a dedicated per port buffer switch architecture, or a big shared physical buffer switch architecture.

Another important factor is the impact of the number of *CFS*' request on the single file server. Preventing Incast needs a sufficient amount of RAM on the hosting file server. A large *WS* packet stores in a RAM. It occupies a space equal to its size. For example, a web server serving 10000 clients having BW capacity of 100Mbits with a 100*ms RTT* requires 1.25 MB (*BDP*) space for each client on the hosting server's RAM —12 GB of RAM. For the same example with 1000Mbits *BW*, each connection needs 8MB of RAM space results into 78GB of RAM. This calculation is valid when all 10000 clients are concurrently requesting data. Figure 15 illustrates these cases comprehensively. For a variable *WS*, as shown in this figure, the product still holds. The *WS*, and *CFS*, both impact system's memory.

In traditional networks, TCP Congestion Window (Cwnd) and Receive Window (Rwnd) are two parameters that are responsible for flow control. These parameters work together to achieve flow control in a TCP network. Our proposed method incorporates the dynamic fair share buffer policy, which allocates equal size of buffer capacity to all concurrent file servers. In summary both congestion window, and receiving window mechanisms are de-coupled from sender and receiver and delegated to a centralized SDS controller (because now we are in SDN environment).

D. SCALABILITY EVALUATION

DCTCP only applicable if the switch supports ECN; it further needs the modification of TCP at sending system end. ICTCP uses the receiver end for Incast mitigation. However, in both of these cases, the mechanism is de-centralized (sender and receiver coordination). The novelty of our work is the centralized control mechanism to mitigate Incast. Both ICTCP and DCTCP cannot scale beyond their allowed number of storage servers. In DCTCP the limit is merely 35 concurrent senders, while in ICTCP, the experiments are done for 48 servers under the same switch. Exceeding that number will cause Incast, which in turn, will not be suitable for a production environment. Even more, in ICTCP, the Incast scenario is only for a single switch, and the connected servers are also belong to the same single switch. In our case, we have incorporated the same switch communication, two switches communication, and more than two switches communication, which is more appealing and practical. Our scheme will not suffer from scaling problem because it is based on SDN. Yes, there is some computing threshold of a single SDN controller, where it will be saturated beyond a maximum number of flows (single point of saturation issue). For that, we suggest to deploy multiple SDS controllers. And now the problem is of the controller's computing scalability, not of throughput; and of course, not our algorithm.

DyFaShBuP provides scalability for hundreds of storage servers. The analysis shows that our scheme achieves a maximum end-to-end I/O throughput for multiple concurrent senders, ranges from a single switch scenario to multiple switches. However, the single point of failure —SDS controller—should be backed-up with a stand-by SDS controller to achieve maximum availability. Moreover, in a large parallel file system where multiple SDS controllers, switches, and storage servers are communicating with each other, predicting a failure is considered as a false-negative alarm [48]. Developing an efficient fault tolerance mechanism needs better perception and understanding of the failure prediction properties.

TABLE 5. Major findings and their description.

Key Point	Description
Small WS	A very small WS introduces a substantial amount of overheads. The file system should provide a data-layout policy not to stripe a data less than some threshold value over N storage servers.
Large WS	BDP is the limiting factor for a large WS. We cannot go beyond BDP. In addition, a large WS for hundreds of thousands of clients imposes pressure on the hosting file server's resources, e.g., memory and CPU.
Number of concurrent clients	For a large number of concurrent clients, a small WS value is not recommended. Our scheme provides a line-rate throughput, and prevents Incast. However, it disrupts the good-put, and faces link-underutilization.
Preventing Incast	The proposed scheme does not limit the number of concurrent clients. It incorporates the dynamic fair-share buffer for end-to-end I/O path, and distributes the policy to all communicating servers/clients.
Buffer dynamics	Dynamic fair-share buffer for end-to-end I/O path calculation and flow control is solely based on the centralized controller policy. The buffer mechanism is decoupled from both receivers and senders. There is no intervention of receiver and sender to control buffer as in DCTCP, and ICTCP.

TABLE 6. Different comparison: Different congestion policies, scope visibility, and deployment comparison among DCTCP, ICTCP, and DyFashBuP.

Technique	Congestion Detection	Policy	Scope Visibility	Scalability	Rate Limiter	Deployment
DCTCP	WS adjustment (WS is coupled with the sender)	Only sending end (sender governs the size of next WS), decentralized buffer coordination by sender	Limited to a single host (flow)	Limited concurrent senders	N/A	Different switches from different vendors affect the overall performance
ICTCP	WS adjustment, (WS is coupled with the receiver)	Only receiving end (receiver governs the size of next WS), decentralized buffer coordination by receiver	Limited to a single host (flow)	Limited	N/A	Multi-hop communication having different switches affect the performance
DyFashBuP	Congestion free (the policy dynamically calculates the fair-share buffer for all concurrent senders. There is no congestion signal to the SDS controller), (WS is de-coupled from both sender and receiver)	Complete end-to-end path (no receiver or sender collaboration, a centralized policy governs the WS)	Global fair-share policy, global view of the flows	Unlimited	Supported (In peak hours, administrator can limit the whole network traffic)	Open standard API are elastic and programmable

E. DISCUSSION

The dynamic buffer policy provides a fair-share end-to-end I/O buffer allocation, and mitigates buffer pressure and Incast. This leads to a minimum latency in a storage network. The target storage network for the proposed scheme is definitely the SDS environment—the new storage paradigm. However, only a dynamic buffer policy is not enough. A proper understanding of striping and spreading size, block size, read and write latency of disk technology, and delay caused by the protocol stack is required to mitigate Incast in a storage end-to-end I/O path. Avoiding striping over a large number of servers may reduce Incast. However, in our method, wider striping does not cause Incast. Wider striping introduces link-underutilization if the *WS* is very small. We recommend a file system policy to stripe a file of some threshold size to avoid link under-utilization. Key major findings are shown in Table 5. Incast techniques' comparison is shown in Table 6.

Unlimited number of control messages and events are generated in any network at high-speed are enough to overload any centralized controller [49]. To guard against controller's failure, a backup policy should be implemented to provide maximum availability. However, multiple SDS controllers lead to a synchronization problem.

For all experiments, we assumed an ideal lossless path. Considering the path loss and protocol overhead, the actual throughput yields less than the available BW. For 1Gbps, the loss and protocol overhead always result between 24 and 27 Mbps.

We have shown the software solution towards Incast. However, an equal emphasis should be on the hardware aspect. Inside the physical network switch, we have two types of buffers: port based buffer, and shared buffer. In a port based buffer, all incoming packets use the per port buffer linked to that port. In the shared buffer architecture, all ports inside a switch use a common shared pool of memory. Per port buffer is fast but expensive; shared buffer is cheap but slow. It is the job of data center administrator to choose the best choice when talking Incast. It depends on the particular application

that a data center uses. Other challenges related to preventing Incast are: number of SDS controllers and their synchronization, different storage topologies, the single controller reliability and resilience, an optimal value for striping and spreading, and a hybrid buffer approach (dedicated vs shared memory).

VII. CONCLUSION

I/O throughput collapse in a storage network depends on the details of the protocol implementation, available BW, link quality, network switches (especially buffer sizes), and system configuration (e.g., the number of servers over which data is stripped). Different techniques have been proposed to mitigate Incast, e.g., reducing TO, limiting number of CFS, and receiver and sender coordination. We have proposed a whole new scheme mitigating Incast in a storage end-to-end I/O path through the centralized SDN inspired way. However, due to the complex connectivity in data centers, additional research is needed in this area to find a solution that fits to all situations.

REFERENCES

[1] D. Nagle, D. Serenyi, and A. Matthews, "The Panasas activescale storage cluster—Delivering scalable high bandwidth storage," in *Proc. ACM/IEEE Conf. Supercomput. (SC)*, Nov. 2004, p. 53.

[2] Y. Chen, R. Griffit, D. Zats, and R. H. Katz, "Understanding TCP incast and its implications for big data workloads," Univ. California Berkeley, Berkeley, CA, USA, Tech. Rep., Jun. 2012, vol. 37, no. 3. [Online]. Available: https://www.usenix.org/publications/login/june-2012/understanding-tcp-incast

[3] H. Song, Y. Yin, X.-H. Sun, R. Thakur, and S. Lang, "Server-side I/O coordination for parallel file systems," in *Proc. Int. Conf. High Perform. Comput., Netw., Storage Anal.*, Nov. 2011, pp. 17:1–17:11.

[4] Y. Zhang and N. Ansari, "On architecture design, congestion notification, TCP incast and power consumption in data centers," *IEEE Commun. Surveys Tuts.*, vol. 15, no. 1, pp. 39–64, 1st Quart., 2013.

[5] A. Phanishayee *et al.*, "Measurement and analysis of TCP throughput collapse in cluster-based storage systems," in *Proc. USENIX Conf. File Storage Technol. (FAST)*, vol. 8, Feb. 2008, pp. 1–14.

[6] K. Phemius and M. Bouet, "Monitoring latency with OpenFlow," in *Proc. 9th Int. IEEE Conf. Netw. Service Manage.*, Oct. 2013, pp. 122–125.

[7] M. Mayer. (Nov. 2006). *Marissa Mayer at Web 2.0*. [Online]. Available: http://glinden.blogspot.com/2006/11/marissa-mayer-at-web-20.html

[8] S. Stefanov. (Dec. 2008). *YSlow 2.0*. [Online]. Available: http://www.slideshare.net/stoyan/yslow-20-presentation

[9] N. McKeown *et al.*, "OpenFlow: Enabling innovation in campus networks," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, Apr. 2008.

[10] D. Kreutz, F. Ramos, P. E. Veríssimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proc. IEEE*, vol. 103, no. 1, pp. 14–76, Jan. 2015.

[11] K. Nichols and V. Jacobson, "A modern AQM is just one piece of the solution to bufferbloat," *ACM Queue Netw.*, vol. 10, no. 5, pp. 20:20–20:34, May 2012.

[12] I. Stefanovici, B. Schroeder, G. O'Shea, and E. Thereska, "sRoute: Treating the storage stack like a network," in *Proc. 14th USENIX Conf. File Storage Technol. (FAST)*, Feb. 2016, pp. 197–212.

[13] A. Munir, G. Baig, S. M. Irteza, I. A. Qazi, A. X. Liu, and F. R. Dogar, "Friends, not foes: Synthesizing existing transport strategies for data center networks," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 4, pp. 491–502, Aug. 2014.

[14] R. Riggio, T. Rasheed, and M. K. Marina, "Poster: Programming software-defined wireless networks," in *Proc. 20th Annu. Int. Conf. Mobile Comput. Netw.*, Sep. 2014, pp. 413–416.

[15] S. Shin, V. Yegneswaran, P. Porras, and G. Gu, "AVANT-GUARD: Scalable and vigilant switch flow management in software-defined networks," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Nov. 2013, pp. 413–424.

[16] K. Chen, H. Zheng, Y. Zhao, and Y. Guo, "Improved solution to TCP incast problem in data center networks," in *Proc. Int. Conf. Cyber-Enabled Distrib. Comput. Knowl. Discovery (CyberC)*, Oct. 2012, pp. 427–434.

[17] M. Alizadeh *et al.*, "Data center TCP (DCTCP)," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 40, no. 4, pp. 63–74, Oct. 2010.

[18] H. Wu, Z. Feng, C. Guo, and Y. Zhang, "ICTCP: Incast congestion control for TCP in data-center networks," *IEEE/ACM Trans. Netw.*, vol. 21, no. 2, pp. 345–358, Apr. 2013.

[19] V. Vasudevan *et al.*, "Safe and effective fine-grained TCP retransmissions for datacenter communication," *SIGCOMM Comput. Commun. Rev.*, vol. 39, no. 4, pp. 303–314, Aug. 2009.

[20] P. Zhang, H. Wang, and S. Cheng, "Shrinking MTU to mitigate TCP incast throughput collapse in data center networks," in *Proc. 3rd Int. Conf. Commun. Mobile Comput. (CCMC)*, Apr. 2011, pp. 126–129.

[21] R. Mittal *et al.*, "TIMELY: RTT-based congestion control for the datacenter," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 4, pp. 537–550, 2015.

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

[23] C. Lee, Y. Nakagawa, K. Hyoudou, S. Kobayashi, O. Shiraki, and T. Shimizu, "Flow-Aware congestion control to improve throughput under TCP incast in datacenter networks," in *Proc. 39th IEEE Conf. Comput. Softw. Appl. (COMPSAC)*, vol. 3, Jul. 2015, pp. 155–162.

[24] J. Hwang, J. Yoo, S. H. Lee, and H. W. Jin, "Scalable congestion control protocol based on SDN in data center networks," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Dec. 2015, pp. 1–6.

[25] M. Ghobadi, S. H. Yeganeh, and Y. Ganjali, "Rethinking end-to-end congestion control in software-defined networks," in *Proc. 11th ACM Workshop Hot Topics Netw. (HotNets-XI)*, New York, NY, USA, Oct. 2012, pp. 61–66.

[26] S. Jouet, C. Perkins, and D. Pezaros, "OTCP: SDN-managed congestion control for data center networks," in *Proc. IEEE/IFIP Symp. Netw. Oper. Manage.*, Apr. 2016, pp. 171–179.

[27] J. Zhang, F. Ren, L. Tang, and C. Lin, "Modeling and solving TCP incast problem in data center networks," *IEEE Trans. Parallel Distrib. Syst.*, vol. 26, no. 2, pp. 478–491, Feb. 2015.

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

[29] Y. Chen, R. Griffith, J. Liu, R. H. Katz, and A. D. Joseph, "Understanding TCP incast throughput collapse in datacenter networks," in *Proc. 1st ACM Workshop Res. Enterprise Netw. (WREN)*, New York, NY, USA, Aug. 2009, pp. 73–82.

[30] T. A. N. Nguyen, S. Gangadhar, and J. P. G. Sterbenz, "Performance evaluation of TCP congestion control algorithms in data center networks," in *Proc. 11th Int. Conf. Future Internet Technol. (CFI)*, New York, NY, USA, Jun. 2016, pp. 21–28.

[31] W. Bai, L. Chen, K. Chen, and H. Wu, "Enabling ECN in multi-service multi-queue data centers," in *Proc. 13th Usenix Conf. Netw. Syst. Design Implement. (NSDI)*, Berkeley, CA, USA, Mar. 2016, pp. 537–549.

[32] H. Wu, J. Ju, G. Lu, C. Guo, Y. Xiong, and Y. Zhang, "Tuning ECN for data center networks," in *Proc. 8th Int. Conf. Emerg. Netw. Exp. Technol. (CoNEXT)*, New York, NY, USA, Jan. 2012, pp. 25–36.

[33] T. Zhang, J. Wang, J. Huang, Y. Huang, J. Chen, and Y. Pan, "Adaptive marking threshold method for delay-sensitive TCP in data center network," *J. Netw. Comput. Appl.*, vol. 61, pp. 222–234, Feb. 2016.

[34] L. Xu, K. Xu, Y. Jiang, F. Ren, and H. Wang, "Throughput optimization of TCP incast congestion control in large-scale datacenter networks," *Comput. Netw.*, vol. 124, pp. 46–60, Sep. 2017.

[35] J. Huang, Y. Huang, J. Wang, and T. He, "Adjusting packet size to mitigate TCP incast in data center networks with cots switches," *IEEE Trans. Cloud Comput.*, to be published.

[36] H. Almasi, H. Rezaei, M. U. Chaudhry, and B. Vamanan. (2018). "Pulser: Fast congestion response using explicit incast notifications for datacenter networks." [Online]. Available: https://arxiv.org/abs/1809.09751

[37] J. Xue, M. U. Chaudhry, B. Vamanan, T. N. Vijaykumar, and M. Thottethodi, "Fast congestion control in RDMA-based datacenter networks," in *Proc. ACM SIGCOMM Conf. Posters Demos*, 2018, pp. 24–26.

[38] Y. Lu and S. Zhu, "SDN-based TCP congestion control in data center networks," in *Proc. IEEE 34th Int. Perform. Comput. Commun. Conf. (IPCCC)*, Dec. 2015, pp. 1–7.

[39] Y. Qin, W. Yang, Y. Ye, and Y. Shi, "Analysis for TCP in data center networks: Outcast and incast," *J. Netw. Comput. Appl.*, vol. 68, pp. 140–150, Jun. 2016.

[40] J. Wang, J. Wen, J. Zhang, Z. Xiong, and Y. Han, "TCP-FIT: An improved TCP algorithm for heterogeneous networks," *J. Netw. Comput. Appl.*, vol. 71, pp. 167–180, Aug. 2016.

[41] A. Bechtolsheim, L. Dale, H. Holbrook, and A. Li, "Why big data needs big buffer switches," Arista, Santa Clara, CA, USA, White Paper, Mar. 2016. [Online]. Available: https://www.arista.com/assets/data/pdf/Whitepapers/BigDataBigBuffers-WP.pdf

[42] J. C. Mogul and R. R. Kompella, "Inferring the network latency requirements of cloud tenants," in *Proc. 15th USENIX Conf. Hot Topics Oper. Syst. (HOTOS)*, Berkeley, CA, USA, May 2015, p. 24.

[43] N. S. Islam, M. Wasi-ur-Rahman, X. Lu, and D. K. Panda, "High performance design for HDFS with byte-addressability of NVM and RDMA," in *Proc. Int. Conf. Supercomput. (ICS)*, New York, NY, USA, Jun. 2016, pp. 8:1–8:14.

[44] N. Liu *et al.*, "On the role of burst buffers in leadership-class storage systems," in *Proc. IEEE 28th Symp. Mass Storage Syst. Technol. (MSST)*, Apr. 2012, pp. 1–11.

[45] B. Behzad, S. Byna, and M. Snir, "Pattern-driven parallel I/O tuning," in *Proc. 10th Workshop Parallel Data Storage (PDSW)*, New York, NY, USA, Nov. 2015, pp. 43–48.

[46] E. Thereska *et al.*, "IOFlow: A software-defined storage architecture," in *Proc. 24th ACM Symp. Operat. Syst. Princ. (SOSP)*, New York, NY, USA, Oct. 2013, pp. 182–196.

[47] H. Song, Y. Yin, Y. Chen, and X.-H. Sun, "Cost-intelligent application-specific data layout optimization for parallel file systems," *Cluster Comput.*, vol. 16, no. 2, pp. 285–298, Jun. 2013.

[48] M. S. Bouguerra, A. Gainaru, and F. Cappello, "Failure prediction: What to do with unpredicted failures," in *Proc. 28th IEEE Int. Symp. Parallel Distrib. Process.*, vol. 2, May 2013. [Online]. Available: https://scholar.google.com.pk/citations?user=6mnFFwYAAAAJ&hl=en&oi=sra

[49] M. Ambrosin, M. Conti, F. De Gaspari, and R. Poovendran, "LineSwitch: Efficiently managing switch flow in software-defined networking while effectively tackling DoS attacks," in *Proc. 10th ACM Symp. Inf., Comput. Commun. Secur.*, Apr. 2015, pp. 639–644.

YAWAR ABBAS BANGASH received the B.S. degree in software engineering from the NWFP University of Engineering and Technology Peshawar, Mardan Campus, in 2008, the M.S. degree in computer engineering from the Wuhan university of technology, Wuhan, China, in 2014, and the Ph.D. degree from the Huazhong University of Science and Technology, China, in 2017. From 2008 to 2012, he was with Huawei Organization Pakistan Ltd., with the Higher Education Commission Project PERN2, and with the Baluchistan Education Foundation on different positions in networking sector. He is currently an Assistant Professor with the Department of Computer Software Engineering, Military College of Signals, National University of Sciences and Technology, Pakistan. His research interests include security in data communications, software-defined networks, wireless sensor networks, and storage systems.

TAUSEEF RANA received the B.Eng. and M.Sc. degrees from London South Bank University, and the Ph.D. degree from The University of Manchester. He is currently serving as an Assistant Professor with the Computer Software Engineering Department, MCS (a constituent college), National University of Sciences and Technology, Pakistan.

HAIDER ABBAS received the M.S. degree in engineering and management of information systems and the Ph.D. degree in information security from the KTH-Royal Institute of Technology, Stockholm, Sweden, in 2006 and 2010, respectively. His professional career consists of activities ranging from R&D and Industry Consultations (Government and Private), through multinational research projects, research fellowships, doctoral studies advisory services, International Journal Editorships, the Conferences/Workshops Chair, an Invited/Keynote Speaker, a Technical Program Committee Member, and a Reviewer for several international journals and conferences. He is currently an Associate Professor with the Department of Information Security, Military College of Signals, National University of Sciences and Technology, Pakistan. He is also a Cyber Security Professional, an Academician, a Researcher, and an Industry Consultant who took professional trainings and certifications from the Massachusetts Institute of Technology, USA; Stockholm University, Sweden; the Stockholm School of Entrepreneurship, Sweden; IBM, USA; and EC-Council. He is also an Adjunct Faculty and a Doctoral Studies Advisor with Florida Institute of Technology, USA.

In recognition of his services to the international research community and excellence in professional standing, he was a recipient of one of the youngest Fellows of The Institution of Engineering and Technology, U.K.; a Fellow of the British Computer Society, U.K., and a Fellow of the Institute of Science and Technology, U.K. He has also been elected to the grade of a Senior Member of the Institute of Electrical and Electronics Engineers, USA.

MUHAMMAD ALI IMRAN (M'03–SM'12) is currently a Professor of wireless communication systems with research interests in self-organized networks, wireless networked control systems, and wireless sensor systems. He also heads the Communications, Sensing and Imaging CSI research group, University of Glasgow. He is also an Affiliate Professor with The University of Oklahoma, USA, and a Visiting Professor with the 5G Innovation Centre, University of Surrey, U.K. He has over 18 years of combined academic and industry experience with several leading roles in multi-million pounds funded projects. He has authored/co-authored over 400 journal and conference publications. He was an Editor of two books and has authored over 15 book chapters. He holds 15 patents. He has successfully supervised over 40 postgraduate students at Doctoral level. He has been a Consultant to international projects and local companies in the area of self-organized networks. He is a Fellow of IET and a Senior Fellow of HEA.

ADNAN AHMED KHAN was born in 1971. He received the degree in telecommunications engineering from the University of Engineering and Technology (UET), Lahore, Pakistan, in 1993, and the M.S. and Ph.D. degrees in computer engineering from the Centre of Advanced Studies in Engineering Islamabad, UET, Taxila, in 2005 and 2009, respectively. He is currently the Head of Computer Software Engineering with the College of Signals, National University of Sciences and Technology, Pakistan. He has published a number of research papers in renowned conferences and journals. His research interests include multi-input multi-output wireless communications systems, software-defined radios, cognitive radios, satellite communications systems, and artificial intelligence.