Floodgate: Taming Incast in Datacenter Networks

Kexin Liu[★], Chen Tian[★*], Qingyue Wang[★], Hao Zheng[★], Peiwen Yu[★],
Wenhao Sun[△], Yonghui Xu[△], Ke Meng[△], Lei Han[△], Jie Fu[△], Wanchun Dou[★], Guihai Chen[★]
[★]*Nanjing University, China* [△]*Huawei, China*

ABSTRACT

Incast occurs frequently in datacenter networks where a large number of senders send data to a single receiver simultaneously, which makes the last hop the network bottleneck. Incast can hurt flows' performance. However, congestion control protocols are not effective at handling incast. One key insight is that it is too late to handle incast packets after they have already piled up at the last hop. Instead, we should avoid incast as early as possible. Inspired by flood control in Hydrologic Engineering, we propose Floodgate, a novel switch-based per-hop flow control to handle incast. Floodgate is compatible with existing congestion control protocols. We integrate it with practical congestion control approaches such as DCQCN, TIMELY, and HPCC. We evaluate Floodgate both in our implementations and large-scale simulations. Compared with state of the art, Floodgate reduces the buffer occupancy by a factor of 6.6×, as well as the queuing delay. Therefore, the average FCT and tail latency are greatly reduced.

CCS CONCEPTS

• **Networks** → **Link-layer protocols**; **Data center networks**; *Programmable networks*;

KEYWORDS

Datacenter, Flow Control, Incast

ACM Reference Format:
Kexin Liu, Chen Tian, Qingyue Wang, Hao Zheng, Peiwen Yu, Wenhao Sun, Yonghui Xu, Ke Meng, Lei Han, Jie Fu, Wanchun Dou, Guihai Chen. 2021. Floodgate: Taming Incast in Datacenter Networks. In *The 17th International Conference on emerging Networking EXperiments and Technologies (CoNEXT '21), December 7–10, 2021, Virtual Event, Germany*. ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3485983.3494854

1 INTRODUCTION

A large number of senders can send data to a single receiver simultaneously, which makes the last hop the network bottleneck. This is called an *incast*. Incast scenarios occur frequently in the datacenter when clients run web search application, Spark-like data-parallel systems, TensorFlow-like machine learning systems, or large-scale storage data backup [13, 15, 17, 40, 44, 47, 53, 55] (§ 2.1).

Incast can hurt flows' performance. First, a large amount of incast traffic could lead to buffer built-up and even overflow, inducing packet drops or Priority-based Flow Control (PFC). Timeout retransmission induced by packet drop could hurt the network throughput and result in long tail latency [4, 52]. PFC could lead to poor application performance due to its congestion-spreading characteristics, *i.e.*, head-of-line (HOL) blocking, routing deadlocks, and PFC pause frame storms [21, 30, 36, 55]. In addition, even if packet loss/PFC does not occur, a large buffer occupancy induced by incast leads to HOL blocking problem thus impairs the performance of flows (§ 2.2).

Congestion control protocols are not effective at handling incast. Recent years have seen numerous congestion control protocols for better application performance. They can be classified into *reactive* and *proactive*. Reactive approaches (*e.g.*, DCTCP [4], DCQCN [55], Timely [38], HPCC [32], PINT [7], Swift [31], and On-Ramp [35]) start at a line rate to send data packets and then react to congestion signals from in-network switches or end-hosts. As for an end-to-end congestion control protocol, it can only take effects when flows last at least for one Round-Trip Time (RTT). Incast traffic may be composed of small flows whose size is smaller than one Bandwidth-Delay-Product (BDP) worth of data. With the rapid growth of link bandwidth (i.e., 100 Gbps is now popular and 200/400 Gbps are on the horizon), more flows can be smaller than one BDP [39], which makes congestion control even less capable in handling incast. Proactive approaches (*e.g.*, ExpressPass [11], Homa [39], NDP [23], pHost [16], and Aeolus [25]) are proposed to allocate bandwidth for data before its transmission. However, they also struggle in handling the first BDP *unscheduled* packets (§ 2.3).

One key insight is that it is too late to handle incast packets after they have already piled up at the last hop. For congestion control protocols, senders will not take action until the in-network congestion signals are carried back. Buffer could have already been saturated by incast traffic. Can we avoid incast as early as possible?

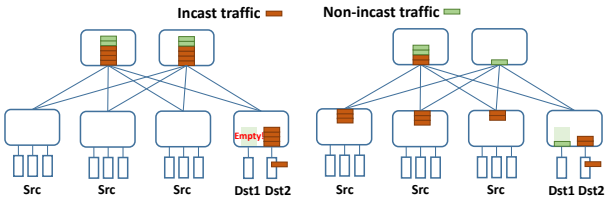
Inspired by flood control in Hydrologic Engineering [22, 50], we propose Floodgate, a novel switch-based per-hop flow control to handle incast. Intuitions are that we could control the transmission of incast traffic via switches as early as possible. Rather than relying on an end-to-end converge-based way to handle incast, it could be more effective to tame its transmission via switches directly (§ 2.4). It poses a set of potential challenges. (i) Incast traffic must be recognized quickly and accurately, without hurting the performance of other non-incast flows. (ii) Determine the volume of traffic the switches should tame to reduce buffer occupancy at the same time ensuring bandwidth utilization. (iii) Hardware resource, *e.g.*, queue and processing power on switches are limited, therefore we must use them appropriately.

Floodgate solves a list of challenges. Floodgate leverages per-dst window to identify incast traffic from other non-incast traffic. Then transmission of incast traffic is tamed via window control before

*Chen Tian is the corresponding author.

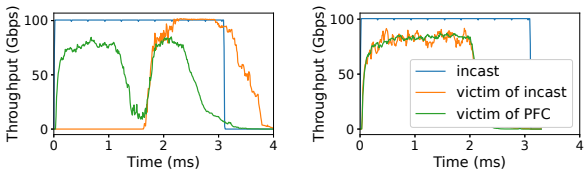
CoNEXT '21, December 7–10, 2021, Virtual Event, Germany

ACM ISBN 978-1-4503-9098-9/21/12…$15.00
https://doi.org/10.1145/3485983.3494854



(a) Incast can hurt the network throughput. (b) Taming the injection of incast can improve the network throughput.

Figure 1: When encounters incast.



(a) DCQCN. (b) DCQCN+Floodgate.

Figure 2: Realtime bandwidth utilization under incast flows mixed with non-incast flows scenarios. Non-incast flows are generated with a load of 0.8 following the Poisson arrival process (§ 6).

it overwhelms the buffer on congestion points. Meanwhile, incast traffic is isolated from non-incast traffic to avoid HOL blocking (§ 3). In addition, hardware resource limitations are also considered (§ 4). Floodgate is compatible with existing congestion control protocols.

We implement Floodgate in Linux hosts with DPDK [14] (§ 5). We integrate it with practical congestion control approaches such as DCQCN [55], TIMELY [38], and HPCC [32]. We evaluate Floodgate both in our implementations and large-scale simulations [41]. The results demonstrate that Floodgate reduces buffer occupancy in incast scenarios (including scenarios where incast flows mixed with non-incast Poisson arrival flows) without bandwidth waste, thus avoiding PFC. Meanwhile, flows' average and tail latency are significantly reduced. Besides, Floodgate does not hurt performances in rarely congestion scenarios, *i.e.*, pure Poisson arrival flows. Compared with state of the art, Floodgate reduces the buffer occupancy by a factor of 6.6×. The average FCT is reduced by 10.1%-98.1% and 99th-tail latency can be 1.1× - 207× lower (§ 6). In addition, hardware feasibility of Floodgate is discussed (§ 7).

2 BACKGROUND AND MOTIVATION

2.1 Incast is not Uncommon

In datacenter networks, incast is not uncommon and can occur intermittently across diverse applications [6, 18, 34, 54]. For web search applications [52, 55], multiple query responses could be returned to the same server simultaneously. In data-parallel systems, *e.g.*, Hadoop [1] and Spark [2], *reduce* phrase aggregates the output data of many *map* servers. Therefore, reduce servers can usually be the bottleneck of the network. In machine-learning systems, the same thing happens when a large number of parameters are aggregated by parameter servers. In ceph-like [49] object-based distributed storage systems, each disk in servers is managed as an independent object storage device (OSD). All OSDs form a full-mesh overlay, where there are two persistent connections (*i.e.*, two directions) between any two OSDs. A single server possesses tens of thousands of connections. Therefore, a large number of requests can be generated by a server simultaneously, which results in corresponding response packets overwhelming the network.

2.2 Incast Hurts Performance

Incast causes packet drop. When incast occurs, the performance of flows can be severely hurt. First, a large amount of incast traffic could lead to buffer built-up and even overflow, inducing packet drops. Timeout retransmission mechanism is usually used to handle packet drops [4]. The timeout value is hard to tune. When the timeout value is large, the network throughput could be hurt thus resulting in long tail latency [4, 52]. While with a small timeout value, senders could retransmit unnecessary, which hurts the network goodput [24].

Incast induces PFC. To avoid timeout retransmission and throughput downgrade induced by packet loss, Priority-based Flow Control (PFC) [27] is deployed in RoCEv2 [5] to ensure a drop-free datacenter network. PFC ensures that the buffer will not overflow by forcing the upstream entity, *i.e.*, switches or hosts' NICs, to pause the transmission of the corresponding ports. PFC is transmitted by the receiver when the ingress queue length exceeds a preset threshold, and the sender is resumed by the receiver when the queue length decreases to a lower threshold. However, PFC pauses traffic at a coarse-grained port or priority-class of ports granularity. Thus head-of-line (HOL) blocking could be induced [55]. It can hurt the network throughput and flows' FCT. Moreover, congestion-spreading characteristics of PFC can lead to routing deadlocks, and PFC pause frame storms [21, 30, 33, 36], *etc.*

Incast causes HOL blocking. In addition, even if packet loss/PFC does not occur, a large buffer occupancy induced by incast leads to HOL blocking problem thus impairs the performance of flows. Figure 1(a) demonstrates a simple example to illustrate the problem. Incast happens when many hosts transmitting traffic to the same destination host Dst_2 simultaneously. Large queue builds up at the core switches and the Top-of-Rack (ToR) switch connected to the destination host. When non-incast flows whose destination is Dst_1 arrive, they can encounter a long queuing delay on the sharing path with incast flows, *i.e.*, the core switches. Non-incast flows suffer HOL blocking caused by incast flows. While the queue on the core switch builds up, the queue on $Dst1$'s last hop is empty, *i.e.*, the bandwidth of $Dst1$ gets wasted.

We conduct a simulation for DCQCN where incast flows are mixed with non-incast Poisson arrival flows to demonstrate the problem of incast and PFC (§ 6.1). Figure 2 demonstrates the real-time throughput. Throughput is monitored on the receiver hosts. We separate Poisson flows sharing the same destination ToR with incast flows (victim flows of incast) from other Poisson flows (victim flows of PFC). Victim flows of incast start to be received by hosts after a long delay, *i.e.*, 1.8ms. It indicates that these flows are HOL blocked by incast traffic and suffer a long queuing delay in network. For other flows, a significant throughput downgrade also occurs between 1ms and 2ms. This is because PFC frame storm happens and causes cascaded pauses on switches. These flows are victim flows of PFC which also suffer HOL blocking.

2.3 Congestion Control Alone is not Effective

Reactive protocols. *Reactive protocols* start at a line rate to send data packets and then react to congestion signals (ECN bits in

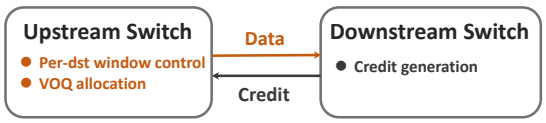


Figure 3: Floodgate framework

DCTCP [4] and DCQCN [55], RTT variation in Timely [38], Swift [31], On-Ramp [35], and INT measurement in HPCC [32] and PINT [7]) from in-network switches or end-hosts. A portion of packets, *i.e.*, generally limited to no more than one Bandwidth-Delay-Product (BDP) worth of data is transmitted before congestion control taking effect. Therefore, congestion control can only take effects when flows at least last for one Round-Trip Time (RTT). Incast traffic may be composed of small flows whose size is smaller than one BDP. With the rapid growth of link bandwidth (i.e., 100 Gbps is now popular and 200/400 Gbps are on the horizon), more flows can be smaller than one BDP worth of data [25, 39]. Therefore, more flows could have been injected into the network even before congestion control starts to kick in. Reactive congestion control becomes less capable of handling incast.

Proactive protocols. There is another category of congestion control called *proactive protocols*. Bandwidth used by data packets is allocated by *token*. A receiver schedules the transmission of token for different flows. When a sender receives a token, it can send a packet, called *scheduled* packet. In ExpressPass [11], packet transmissions are totally scheduled. However, it costs the sender one RTT to notify the receiver before receiving tokens. It wastes bandwidth and hurts the performance of small flows [25]. A natural way to fix the problem is to send *unscheduled* packets at the first RTT. In Homa [39], NDP [23], pHost [16], and Aeolus [25], a portion of unscheduled packets, *i.e.*, usually one BDP, can be sent when a new flow arrives. Because of the involution of unscheduled packets which can not be controlled in advance, proactive protocols also face incast problems.

2.4 Intuition

In Hydrologic Engineering, to control flood, adjustable gates are used to control water flow in flood barriers, reservoirs, or rivers, working as a part of flood bypass systems [22, 50]. Therefore, the water levels in downstream main rivers or canal channels that aggregate upstream traffic can be lowered, thus preventing water flooding.

We are inspired by flood control in Hydrologic Engineering. Instead of handling incast at the last hop, the traffic pressure should be reduced as early as possible. Intuitions are that we could control the transmission of incast traffic via switches directly. Every switch takes its responsibility to tame and store a portion of incast traffic. Hence, incast traffic is shared among switches it passing through. Thus buffer occupancy on the network bottleneck could be significantly reduced. Figure 1(b) illustrates our naïve idea. Incast traffic is tamed by switches it passing through. All switches take their responsibilities to buffer in-network traffic passing through. Thus, the buffer on the aggregation point of incast traffic, *i.e.*, core switches and the destination ToR, is not saturated aggressively. For that buffer occupancy on core switches is reduced, non-incast flows can pass through it quickly, therefore the bandwidth of non-incast flows' last hop is well utilized. Figure 2(b) demonstrates that when Floodgate, a protocol adapted from the intuition, is applied, throughput of flows is not hurt either by incast. At the same time, PFCs are not triggered.

3 FLOODGATE OVERVIEW

3.1 Basic Idea

Intuitions are that by controlling the transmission of incast traffic via switches, maximum buffer occupancy on the last hop could be reduced significantly. The challenge is how to recognize incast traffic quickly and accurately, without hurting the performance of other non-incast flows. By solving the challenge, we propose our strawman Floodgate design, a switch-based hop-by-hop flow control. The basic idea is to leverage per-dst window on switches to identify and control incast traffic. Furthermore, incast traffic should be isolated from non-incast traffic by switches, to make sure that non-incast traffic will not be HOL blocked by incast traffic and could pass through the switch smoothly.

3.2 A Strawman Design

Incast identification. Floodgate's switch uses a per-dst sending window to control the transmission of traffic for each receiver host. In general, with the load-balanced network, flows sharing the same source and destination pair pass through the same congestion point in the network (or different points with the same degree of congestion, see online Appendix [41] for more details). Therefore, per-dst granularity is enough to handle traffic traversing different congestion separately. When Floodgate's switch receives a data packet from an ingress port, it can be forwarded to its downstream switch only when there is sending window left. When a data packet is forwarded, the corresponding sending window is decreased by one. At the same time, the switch immediately generates a credit packet to its upstream switch. The meaning behind this is that when a packet is successfully transmitted, there could be more free space for another packet. Correspondingly, when a switch receives a credit packet from its downstream switch, the sending window of the corresponding destination host is incremented by one.

The window is a probing mechanism to sniff whether the flow is encountering an incast collapse or not. For non-incast flows, the sending window does nothing on their transmission. Data packets are always forwarded successfully by the downstream switch. Hence, credit packets are received in time and the sending window is plenty every time a data packet arrives. For incast flows, the returning rate of credits is limited by the last hop ToR, *i.e.*, the network bottleneck. As a result, sending window can not be recovered in time. Therefore, the transmission of incast traffic is controlled. In this way, the per-dst sending window is a traffic control mechanism as well as an incast identification mechanism. In particular, when data packets arrive at their last hop ToR, it is in vain to buffer them because it does nothing to the network state. Hence, Floodgate's last hop ToR does not maintain a sending window, and the servers do not need to generate credit packets for the last hop ToR connected to them.

The size of sending window indicates how strictly a destination will be recognized as encountering an incast. Given that per-hop credit is used, the sending window can be initialized with a small

value, *i.e.*, $m * BDP_{nextHop}$, where $BDP_{nextHop}$ is the bandwidth-delay product between the switch itself and its next-hop downstream switch, and m is a parameter that stands for how aggressively Floodgate recognizes a flow as encountering an incast.

Incast isolation. After a flow is identified as encountering an incast, following data packets sharing the same destination host will not be forwarded until the sending window is recovered. A dedicated queue should be used to store these data packets to avoid blocking other non-incast traffic. Rather than using physical egress output queue, Floodgate leverages Virtual Output Queue (VOQ) to store incast traffic. Traditionally, a VOQ is a collection of buffers that receive and store traffic destined for one output queue on one egress port [10, 12, 26, 29]. Instead of directly sending packets to the egress queue, a VOQ does not transmit packets until the egress port has the resources to forward the traffic. In Floodgate, data packets transmitting to the same receiver hosts are allocated with a dedicated VOQ. It is aimed at isolating traffic passing through different congestion points.

3.3 Practical Challenges

The strawman design gives a leg up with incast scenarios where congestion control protocols struggle with. However, it does not take switches' cost and network overhead into consideration.

Per-packet credit is costly. In our strawman design, an ideal way to update sending window is adopted. When a data packet is forwarded to the switches' egress queue, a credit is sent back to its upstream switch immediately. However, it is costly for switches to generate credit packets at a high rate, for that it consumes the pps (packets per seconds) of switches. Besides, per-packet credit consumes network bandwidth.

The number of VOQs is limited. Ideally, a switch can allocate a dedicated VOQ for each destination host in advance. However, there can be tens of thousands of hosts in the network, and the hardware resource for switches is limited. Therefore, the usage of VOQs should be more conservative.

4 FLOODGATE DESIGN

In this section, we solve practical challenges on switches' hardware resource limitations and present our final design of Floodgate. Besides, we answer the question that how much volume of traffic switches should tame to ensure bandwidth utilization when reduces buffer occupancy. Floodgate is a switch-based per-hop flow control protocol. Floodgate's design is composed of downstream switch parts and upstream switch parts, as demonstrated in Figure 3. The downstream switch generates credit packets to update the per-dst window of the upstream switch. The upstream switch maintains per-dst window to identify and tame the transmission of incast traffic. Meanwhile, it should allocate VOQs for recognized incast traffic to isolate them from non-incast traffic.

4.1 Downstream Switch Algorithm

Transmit credit packet. Instead of generating credit at a per-packet granularity, Floodgate aggregates credit packets. A Floodgate's switch leverages a timer T for each ingress port. Timer T records the elapsed time since last sent back credits. It determines the credit generation granularity. At the same time, Floodgate's switch records the number of packets that have already been forwarded but have not been returned with credits for each destination host from an ingress port. When the elapsed time reaches the preset value, a credit packet is sent back from the ingress port to the corresponding upstream switch, containing a pair of destination host's IP and the number of credits, *i.e.*, $< destination\_IP, credits >$. And the timer is reset. When there are no credits to be transmitted (*e.g.*, no corresponding traffic has been forwarded), the transmission of the credit packet is skipped.

Delay transmission of credit. When the elapsed time since the last sent credit packets reaches the preset value, a credit packet is transmitted to the corresponding upstream switch. The assumption behind this is that when data packets are successfully forwarded to the next hop, there could be equivalent free space for the following data packets. However, when incast happens, VOQ can build up. The transmission of the corresponding credit could be delayed because it is burdensome for the VOQ to absorb more packets. Floodgate proposes *delayCredit* mechanism. Only when the VOQ length of the corresponding destination does not exceed threshold $thre_{credit}$ will Floodgate's switch generates credit packets. Otherwise, the credit transmission is skipped, and the timer is reset. It avoids unnecessarily buffer buildup.

4.2 Upstream Switch Algorithm

Initialize sending window. Because of credit aggregation, the switch's per-dst sending window for each destination host can not be simply initialized to $m * BDP_{nextHop}$ like the strawman. Instead, as the timer T increases, the initial value of sending window should also be increased to make sure that bandwidth will not get wasted. To utilize the network, the switch's sending window is initialized to $BDP_{nextHop} + C_{out} * T$, where T is the transmission interval of aggregated credit packet, and C_{out} is the switch's egress port bandwidth. Recall that $BDP_{nextHop}$ is the BDP between the switch itself and its next-hop downstream switch (§ 3.2). Hence, when no congestion occurs, the sending window will not be used up before a credit packet is received.

Update per-dst window. When a data packet is forwarded to an egress queue, the sending window for the destination hosts is decreased by one. When a switch receives a credit packet from its upstream, it parses the destination hosts and the number of credits in it. The sending window of the corresponding destination host is increased by the number of credits. At the same time, the transmission of data packets in the corresponding VOQ is triggered.

Allocate VOQs for incast flows dynamically. In Floodgate, VOQ resources are pre-allocated statically, and are allocated for incast traffic dynamically. When a data packet is received, the switch first checks whether the data's destination host has already been allocated with a VOQ. If so, it indicates that the corresponding destination is encountering an incast. Therefore, the data packet is pushed into the corresponding VOQ. Otherwise, the switch checks whether there is sending window left. If the sending window is adequate, the data packet is forwarded to the egress queue directly. Otherwise, the switch chooses an empty VOQ (if possible) according to the VOQ bitmap (§ 7.2) and pushes the data packet into it. Today's datacenter switches can support up to thousands of VOQs [10, 12, 26, 29]. For that only incast traffic uses VOQs, and scenarios where thousands of incast occur simultaneously can be rare, VOQs will not be used up in most scenarios. For robustness, we still consider this rare scenario. When the usage of VOQ reaches the upper bound, a hash function, *e.g.*, Cyclic redundancy check (CRC) [20, 42] is used

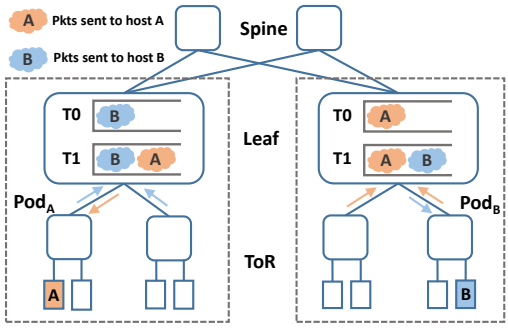


Figure 4: Deadlock problem.

to get an appropriate VOQ to enqueue the data packet by using the destination IP in its header as the hash key. In our evaluation, we found that dozens of VOQs are basically enough § 6.1.

VOQ grouping. Given that the number of VOQs is limited, corner cases could occur where packets with different destination hosts queue in the same VOQ. Without specific handling, a deadlock could occur. Figure 4 demonstrates leaf switches' VOQ queuing state when a deadlock occurs. Hosts belonging to pod A transmit packets to host B, while hosts belonging to pod B transmit packets to host A. At $T0$, a portion of packets whose destination is host B is forwarded by leaf switch A, while the left packets are buffered in leaf switch A because of sending window control. It is the same case for leaf switch B. At $T1$, packets whose destination is host B arrive at leaf switch B, queuing behind packets waiting for credits whose destination is host A. However, these packets whose destination is host A would not receive credits. Because leaf switch A would not send credits to its downstream core switch. For that data whose destination is host A is also blocked in leaf switch A by data with destination host B, which waits for credits from its upstream. Consequently, the core switch would not send credits to its downstream leaf switch B. Briefly speaking, *hold and wait* happens, *i.e.*, packets occupy the buffer in the leaf switch B and wait for credits from upstream switches, while credits cannot be received because of occupied buffer in the leaf switch A. They form a cyclic dependency, therefore results in a deadlock.

To fix the deadlock problem, Floodgate breaks *hold and wait* condition. The root cause of *hold and wait* is that upstream and downstream traffic of leaf switches are mixed into the same VOQ. Downstream traffic for a leaf switch stands for data whose destination host belongs to its pod, while upstream traffic stands for data whose destination belongs to other pods. When these two kinds of traffic are isolated into two dedicated VOQs, packets will not be held in the buffer endlessly. Downstream traffic can be forwarded smoothly and then the corresponding credits can be sent back to its upstream switch. Generally speaking, in Floodgate's middle-layer switches which could forward downstream/upstream traffic simultaneously, *i.e.*, leaf in three-tier topology, VOQs are classified into two groups. A portion of VOQs are reserved for downstream traffic, the left are for upstream traffic.

4.3 Miscellaneous Detailed Designs

Handling (rare) packet loss. Floodgate significantly reduces the buffer occupancy; thus, buffer overflow could be rare. However, there could be corner cases where data/credit loss is induced by configuration errors in switches or link failure. This can result in window vanishing which can hurt the network throughput. Floodgate uses a specific incremental sequence number (PSN) between Floodgate's switches to detect packet loss and achieve fast recovery. Floodgate's switch maintains the PSN of the next send data packet and last sent/received data/credit packet for each pair of ingress port and destination host. The credit packet will pick back the last PSN of the corresponding destination's data packets of this egress port. The difference between $next\_send\_data$ and $last\_recv\_credit$ is the inflight packets and the remaining sending window is equal to $intial_{win} - inflight$. And a relatively large timeout is used to avoid the loss of the last continuous data/credit packets. If the elapsed time since last receiving the credits from its downstream switch reaches the timeout value, the switch generates a *switchSYN* packet to the downstream switch. When *switchSYN* is received, the switch generates credits with PSN to assist its upstream switch to handle packet loss.

Hosts' support (optional). Floodgate can reallocate the in-network traffic from the congestion point to other points, thus relieving the buffer pressure on the congestion point. However, it can not reduce the total volume of in-network traffic. Only with the help of hosts can Floodgate solves the dilemma where the in-network traffic injection overwhelms the overall buffering ability of switches. We propose a one-hop *per-dst PAUSE* mechanism to reduce the in-network traffic. When a Floodgate's ToR receives a data packet whose VOQ queue length exceeds a threshold $thre_{off}$, a special PAUSE frame, *i.e.*, called *dstPause*, is sent back to source hosts, piggybacking the corresponding destination IP. Meanwhile, NICs maintain per-dst queues to pause the corresponding traffic. When a host receives a *dstPause*, it parses the frame and pauses the flow whose destination matches. When the VOQ queue length goes beneath a threshold $thre_{on}$, a RESUME frame called *dstResume* is sent back to hosts to resume the transmission of the corresponding traffic. Note that the dstPause/dstResume frame is only sent by first hop ToR to source hosts connected to them, therefore no deadlock or pause frame storm would happen. Meanwhile, they operates at a per-dst granularity, no HOL blocking on hosts would occur. $thre_{off}$ and $thre_{on}$ can be set to a relatively small value, *e.g.*, one-hop BDP. We add *per-dst PAUSE* in strawman Floodgate to explore its ideal performance.

5 TESTBED EXPERIMENTS

5.1 Implementation

We prototyped Floodgate Software Switch (FSW) on commodity x86 servers based on BESS [8], a scalable software switch architecture that uses DPDK [14] to accelerate data plane packet processing. As is shown in Figure 5, FSW is divided into a control plane and a data plane. Among them, the control plane is responsible for the management of the Poll-Mode Driver (PMD) Port, the configuration of VOQ queues, the packet routing, and the pipeline monitoring for queuing length information. The data plane is composed of the following three modules:

- **Merge** module aggregates packets from multiple input ports to allow for serial processing of all packets through a single pipeline. This is compatible with the current switch design [9], and the pipeline design makes it easy to share switch resources such as flow tables and VOQ queues.

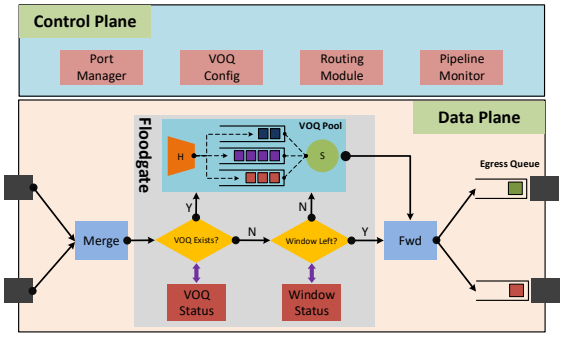
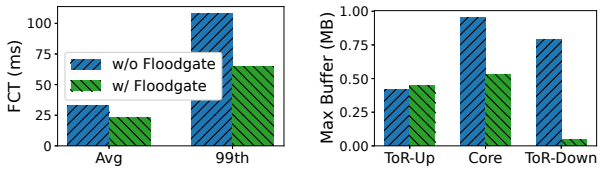


Figure 5: Overview of Floodgate software switch architecture. H denotes a hash module, which is used to select a VOQ queue by hashing the destination IP address. S denotes a queue scheduler, responsible for moving packets from VOQ queues to the next module according to per-dst window status.



(a) FCT of non-incast flows. (b) Buffer occupancy.

Figure 6: Performance of testbed experiments under incast-mix scenarios. ToR-Up denotes ToRs' all ports connected to their downstream switches, *i.e.*, the first hop of packets. Accordingly, ToR-Down denotes the ToRs' all ports connected to hosts, *i.e.*, the last hop of packets.

- **Floodgate** module implements the main function of the Floodgate switch such as per-dst window control and VOQ queue management (§ 4). There is a scheduler that scans multiple VOQs in a round-robin (RR) fashion (see details in § 7.2) and forwards the packet if there is enough window for the dst IP of the packet.
- **Forwarding (Fwd)** module is responsible for the network layer forwarding function, which matches and forwards packets to the queue of the corresponding egress port.

In the experiment, we use the BESS script to install static forwarding rules from the control plane of FSW. We deployed FSW on multiple servers. Table 1 lists the hardware and software configurations in our experiment environments.

5.2 Testbed Evaluation

Topology. The testbed topology consists of one core switch, and three ToR switches. Each ToR switch is connected with two servers via 10Gbps links, and three ToR switches are connected to the Core switch via 20Gbps links (rate of the NIC interface is limited from 40Gbps to 20Gbps by the BESS scheduler's rate limit function). The base BDP is 45KB.

Floodgate improves flows' performance. We evaluate Floodgate when incast flows are mixed with non-incast flows. Four source hosts transmit cross-rack BDP-sized flows to one destination host simultaneously to generate incast traffic patterns. And non-incast Poisson arrival flows are transmitted among hosts except for the destination host of incast. Following [32, 35], a per-flow sending window on hosts is added to emulate the first-RTT actions of DCQCN (§ 6). Figure 6(a) demonstrates the FCT performance of w/o and w/ Floodgate. Without Floodgate, non-incast flows are HOL blocked by incast flows, suffering a large queuing delay. By leveraging Floodgate, the average FCT is reduced by 30.6%, and the 99th-tail latency is reduced by by a factor of 1.6×.

Floodgate reduces the buffer occupancy. Figure 6(b) shows the maximum buffer occupancy of w/o and w/ Floodgate. Without Floodgate, data packets are mainly buffered on the core switch and destination ToR, *i.e.*, ToR-Down. By leveraging Floodgate, incast traffic is tamed via per-dst window, therefore the buffer on ToR-Up is slightly larger. Meanwhile, the maximum buffer occupancy on ToR-Down and core switches is reduced by a factor of 17.2× and 1.8×, respectively.

Table 1: Experiment environments.

Hardware	CPU	Intel(R) Xeon(R) CPU E5-2650 v4 @ 2.20GHz
	NIC	Intel X710 10Gbps Intel XL710 40Gbps
Software	DPDK	19.11.4

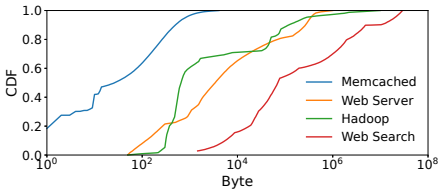


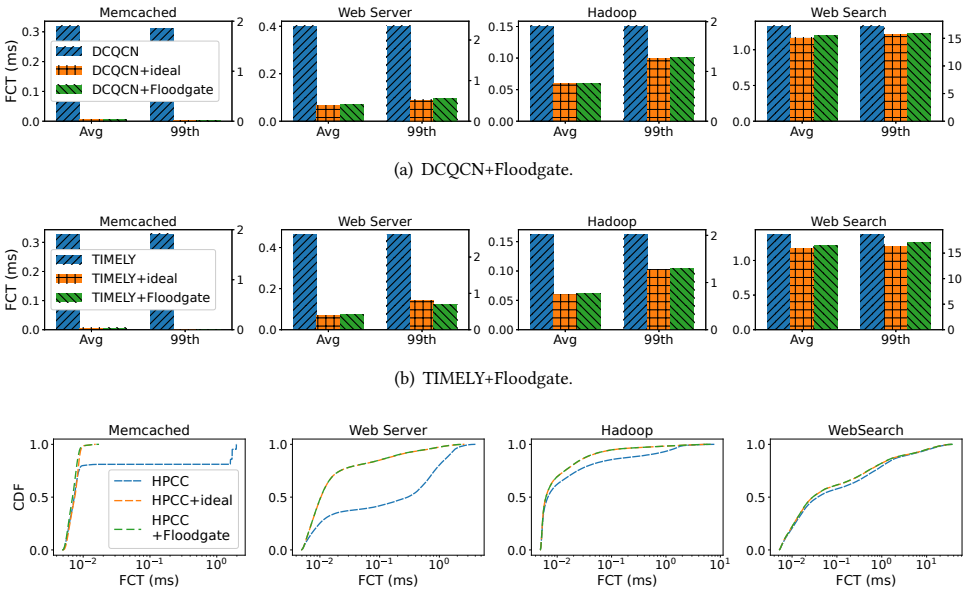
Figure 7: Flow size distribution of typical workloads.

6 SIMULATION EVALUATION

In this section, we use large-scale NS3 simulations to evaluate Floodgate [41]. We integrate strawman ideal Floodgate (ideal for short) and final practical Floodgate (Floodgate for short) with congestion control approaches such as DCQCN [55], TIMELY [38], and HPCC [32]. The author's contributed simulation codes are used in our evaluations [3]. Following [32, 35], a per-flow sending window on hosts is added to DCQCN, TIMELY, and HPCC, limiting the in-flight packets of a flow. Besides performance comparison, we also validate major design points and parameter selection. For space limitation, some results of TIMELY and HPCC are omitted given that similar trends as in DCQCN are observed. *Unless otherwise specified, we use DCQCN as our main comparison protocol.*

In summary, under scenarios where incast flows are mixed with non-incast Poisson arrival flows, Floodgate tames the transmission of incast flows via switches thus reducing buffer occupancy significantly and eliminating PFC, without bandwidth waste. In addition, incast flows are isolated from non-incast flows, therefore further speeding up non-incast flows' FCT. Under pure Poisson scenarios, Floodgate does not hurt flows' performance (Appendix A.2). Although Floodgate can not achieve as small buffer occupancy as the ideal design, it can achieve significant improvements compared to state-of-the-art congestion control protocols as well.

Topology. The topology we used is a 2-level leaf-spine non-blocking network that contains 4 core switches, 10 ToRs, and 160 hosts, *i.e.*, 16 hosts per rack (similar to the topology used in [39]). Each ToR connects its hosts and cores via 100/400Gbps links, respectively. The propagation delay for each hop is 600ns. The base RTT is



(a) DCQCN+Floodgate.

(b) TIMELY+Floodgate.

(c) HPCC+Floodgate.

Figure 8: Average and 99th-tail FCT of Poisson flows under incastmix scenarios.

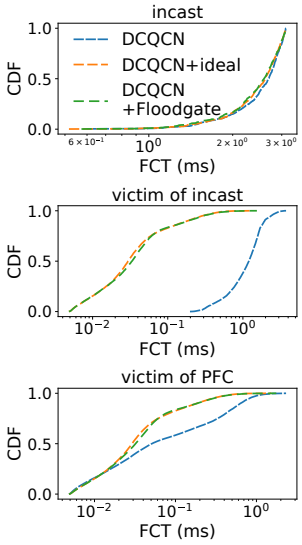


Figure 9: FCT of flows under Web Server incastmix scenario.

5.1μs, and the base BDP is 64KB. Besides, we also use a 3-tier fat tree topology (k=8), consisting of 32 edge switches, 32 aggregation switches, 16 core switches, and 128 hosts (4 hosts per edge, 16 hosts per pod) to evaluate Floodgate's performance [11]. Without special instructions, we use the 2-tier topology.

Workloads. Unless otherwise specified, we generate non-incast flows following a Poisson arrival process with a load of 0.8 and periodic incast flows whose size is between 30 MTU and 40 MTU with a load of 0.5 (the average load of the incast destination host). For Poisson arrival flows, we use four workloads, Memcached [39], Web Server [43], Hadoop [43], and Web Search [4]. The flow size distributions are shown in Figure 7. Memcached is composed of small flows, where most of the flows are smaller than 1KB. The left three workloads are large flows mixed with small flows where a small ratio of large flows dominates the average flow size.

Parameters. We have a set of default settings in evaluations. Here, credit timer $T = 10\mu$s, $delayCredit$ threshold $thre_{credt} = 10$BDP is set for Floodgate. And $m = 1.5$ is set for the ideal design. Besides, the maximum number of VOQs can be used is set to 100, but Floodgate only uses one VOQ in most cases. A dedicated subsection discusses why these values are used (§6.5). For DCQCN/TIMELY/HPCC, the recommended parameter settings are used. Dynamic PFC threshold is used and $\alpha = 2$. The switch buffer capacity is 20MB.

Metrics. We have three major performance metrics: (i) maximum switch buffer usage, (ii) average/99th-tail FCTs, and (iii) number of triggered PFC. We also monitor queuing time and maximum port buffer usage at a per-hop granularity to analyze the buffer reallocation that Floodgate brings about. We run experiments ten times. Results show similar trends. The standard deviation is omitted given that it is relatively small.

6.1 Simulations Under Incastmix Scenarios

Floodgate improves flows' performance. Figure 8 shows the average and 99th-tail FCTs of Poisson flows across different workloads under incastmix scenarios. (Floodgate does not affect the FCT of incast flows, for space limitation, we leave more results of incast flows' FCT in Appendix A.1.) Floodgate reduces the average FCTs by 10.1%-98.1%. And the 99th-tail latency is 1.1× - 207× lower. The improvement comes from significantly reducing buffer occupancy to eliminate PFC. Meanwhile, non-incast flows are not HOL blocked by incast flows anymore via VOQ isolating.

(i) DCQCN. By leveraging Floodgate, the average and tail FCT of Memcached and Web Server are significantly reduced. Memcached workload is mostly composed of flows smaller than one MTU, which can be greatly hurt by a long queuing delay. By leveraging Floodgate, flows' queuing delay is greatly reduced. Therefore, the performance of Memcached is significantly improved. For Web Server, without Floodgate, the throughput of flows are greatly hurt by incast, as well as PFC pause frame storm (as shown in Figure 2). To dig into the performance improvement for victim flows of incast/PFC, Figure 9 shows the FCT distribution of them, respectively. By leveraging Floodgate, flows do not suffer HOL blocking caused by incast and PFC. Therefore, the performance of victim flows of incast/PFC is significantly improved, without compromising the performance of incast flows. For Hadoop and Web Search workloads, the improvement is less obvious. This is because non-victimized large flows dominate the average/tail FCT. However, the FCT of victim flows of incast are greatly reduced (see Figure 22 in § A.1).

(ii) TIMELY. Floodgate's improvement for TIMELY is similar to that for DCQCN. Note that the tail latency of TIMELY+ideal under Web Server is a little larger than TIMELY+Floodgate. This is because

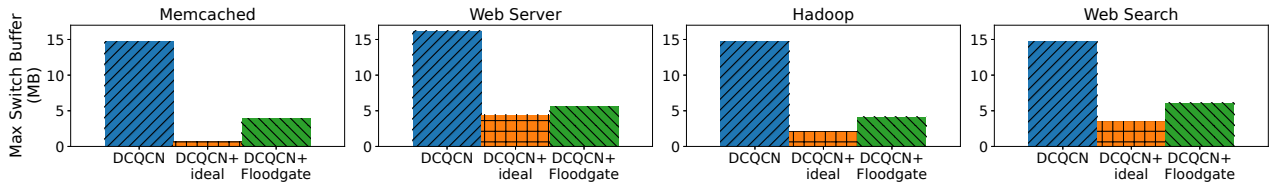


Figure 10: Maximum buffer occupancy.

Table 2: PFC triggered time of DCQCN.

	Memcached	Web Server	Hadoop	Web Search
Host (us)	0	10594.0	0	0
ToR (us)	0	3999.5	0	0
Core (us)	6253	6578.9	6508.6	37152.16

credit packets of ideal design consume more bandwidth (see in Figure 18).

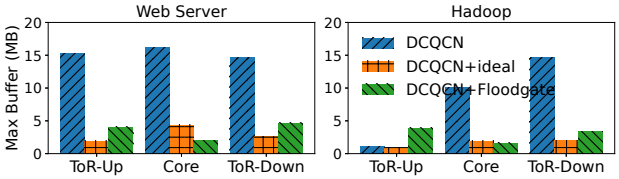
(iii) HPCC. To make it more clear, FCT distribution is used to present the improvement made on HPCC. Similarly, the improvement on Memcached and Web Server workloads is more significant.

Floodgate reduces the buffer occupancy. Figure 10 shows the maximum buffer occupancy across all four workloads. By leveraging Floodgate, the maximum buffer occupancy is reduced 2.4× - 3.7×. This is because Floodgate tames the transmission of incast flows by leveraging per-dst window control. Each switch takes its responsibility to hold back a portion of in-network incast traffic, relieving the pressure on the network bottleneck, *i.e.*, the last hop destination ToR. A by-product of Floodgate is that in-network traffic is distributed more uniformly. As expected, the ideal approach can reduce the buffer occupancy more effectively, because the initial sending window can be set to a smaller value. It has been proved that the maximum buffer occupancy of original DCQCN is proportional to the number of flows, while with Floodgate, the value is decreased to be proportional to the number of core switches, which can reduce the buffer occupancy by an order of magnitude (see more details in online Appendix [41]).

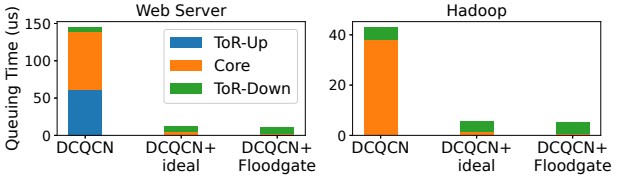
The buffer occupancy is made up of two parts, *i.e.*, the buffer used by incast flows and non-incast Poisson flows. This is the reason why buffer occupancy of Web Server, Hadoop, and Web search is a little larger than Memcached, where non-incast large flows in these workloads occupy a portion of buffer and Floodgate rarely affects them. This is the part left for congestion control protocols.

VOQ usage. We monitor the maximum number of VOQ usage and find that at most one VOQ is used simultaneously by Floodgate's switch. Therefore, no HOL blocking occurs because the number of VOQ used does not exceed the upper bound. And the VOQ is exactly used to buffer packets whose destination host is suffering incast. It means the incast flows are identified accurately, with no non-incast flows are wrongly recognized as incast flows.

PFC triggered time. DCQCN triggers PFC while Floodgate does not, and Table 2 shows the total PFC triggered time of DCQCN. Under all four workloads, PFC is triggered by incast. Particularly, under Web Server workload, core switches, source ToRs, and hosts are all paused by PFCs. In the beginning, the destination ToR is the most congested point, and it sends back PFCs to its upstream core switch. The core switches' egress ports connected to the incast destination ToR are paused. Meanwhile, incast flows are still flooding into core switches. Therefore, the buffer occupancy on



(a) Traffic reallocation: buffer occupancy on each hop switches.

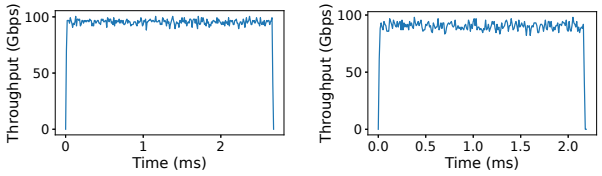


(b) Split up queuing time.

Figure 11: Traffic reallocation and queuing time analysis. ToR-Up denotes ToRs' all ports connected to their downstream switches, *i.e.*, the first hop of packets. Accordingly, ToR-Down denotes the ToRs' all ports connected to hosts, *i.e.*, the last hop of packets.

core switches is increasing until PFCs are sent back to its upstream source ToR switch. It is the same case for source ToRs sending back PFCs to their connected hosts. It is called PFC pause frame storm [21]. PFC pause frame storm can spread the incast congestion to the whole network, thus cause damage on network throughput and FCT, which is consistent with throughput and FCT performance demonstrated in Figure 2 and Figure 9.

Traffic reallocation and queuing time analysis. In this part, we dig into how Floodgate reallocates in-network traffic and correspondingly affects packets queuing time among different hops. Figure 11(a) demonstrates the maximum buffer occupancy among different hops. For DCQCN, under Hadoop workload, because of the non-blocking topology, the buffer occupancy on ToR-Up ports, *i.e.*, the first hop of packets, is nearly zero. The maximum buffer occupancy on ToR-Down ports, *i.e.*, the last hop of packets, is the largest among all kinds, followed by buffer occupancy on core switches. This is because ToR-Down ports and core switches are the aggregation point of incast traffic. Particularly, under Web Server workload, the maximum buffer occupancy on ToR-Up ports is also large because of the PFC frame storm. Floodgate's sending window control reallocates the in-network traffic and makes them distributed more uniformly. The buffer pressure put on the incast aggregation points is greatly reduced. Therefore, the buffer occupancy on core and ToR-Down ports is reduced significantly. Meanwhile, a relatively larger buffer occupancy on ToR-Up ports is observed, for that incast traffic is firstly tamed by source ToRs to avoid injecting into the downstream network aggressively. Ideal



(a) 5% packet loss rate. (b) 10% packet loss rate.

Figure 12: Throughput when packet loss occurs.

Floodgate design leverages *per-dst PAUSE* mechanism to reduce the in-network traffic, therefore the buffer occupancy on ToR-Up ports is smaller than Floodgate.

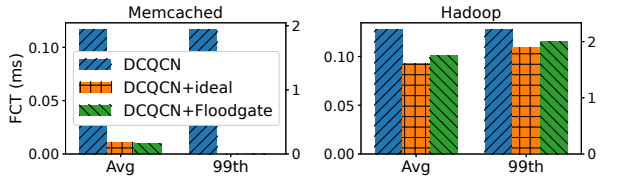
Figure 11(b) demonstrates the per-hop average queuing time of non-incast flows respectively. For DCQCN, under Web Server workload, packets spend much time on ToR-Up ports because of PFC frame storm. Under Hadoop workload, the time spent on core switches constitutes most of the queuing time because of HOL blocking caused by incast flows. When Floodgate is applied, the queuing time among each hop is greatly reduced. A slightly larger buffer occupancy on ToR-Up ports does not affect flows' queuing time, for that incast traffic is isolated via VOQs.

6.2 Robustness of Floodgate

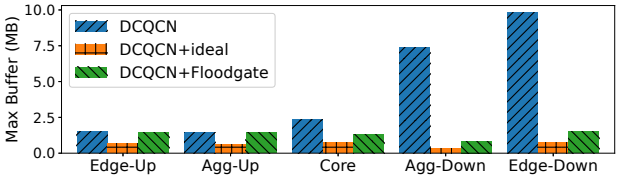
Handling packet loss. To test the robustness of Floodgate when encountering packet loss, we manufactured packet drops. Figure 12 shows the result. Under packet loss rate 5%, no obvious effect on throughput has been observed. Under packet loss rate 10%, throughput fluctuates within a small range. It indicates that the sending window of Floodgate's switch can be recovered quickly after packet loss happens.

Three-tier topology. To explore Floodgate's performance under different topologies, an 8-ary fat tree topology is used. Floodgate reduces the average FCT and tail latency of Poisson flows significantly, especially for Memcached, as shown in Figure 13(a). The effectiveness of Floodgate is slightly smaller than in the two-tier topology (§ 6.1). The main reason is that in this fat tree topology, the number of switches is far larger while the number of hosts is smaller than the two-tier topology. In addition, non-incast flows whose destination belongs to the same rack with incast flows suffer HOL blocking, for that they share the same egress port with incast on aggregation switches. Given that number of hosts per rack is reduced from 16 to 4, victim flows of incast become less.

When the number of ToR scales up. Figure 14 depicts the results when the number of ToR scales up. Pure incast traffic is generated where all hosts (except for the destination host) participate in transmitting one flow to the same destination host. The flow size ranges from 30 MTU to 40 MTU. For DCQCN, the buffer occupancy on ToR-Down ports increases quickly when the number of ToRs increases, at a rate proportional to the number of flows. When the number of ToRs reaches 20, PFC is triggered and the buffer occupancy reaches its maximum value. Floodgate is more robust to handle large-scale topology. The buffer occupancy of Floodgate remains stable when the number of ToR increases. One may wonder why the buffer occupancy of core switches remains stable, as they can receive data packets from more ToR when the number of ToR increases. This is because Floodgate benefits from the *delayCredit* mechanism. Core



(a) Average and 99th-tail FCT of Poisson flows.



(b) Buffer occupancy on each hop switches under Hadoop workload.

Figure 13: Performance in an 8-ary fat tree topology. Edge-Up/Agg-Up denotes ports connected to their upstream switch; Edge-Down/Agg-Down denotes ports connected to hosts /their downstream switch.

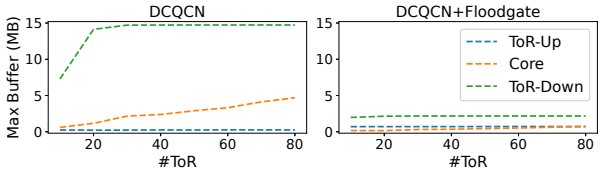


Figure 14: When number of ToR switches scales up.

switches are delayed to send back credits when there are plenty of packets buffered in VOQs.

6.3 Design Choice

Per-dst PAUSE is more robust to successive incast. Generally, source ToRs are far less congested than the last hop destination ToR. Nevertheless, we now consider an extremely corner scenario where incast traffic is generated several times successively, *i.e.*, each host transmits an incast flow to one destination host and then another. Figure 15 demonstrates the results when several successive incast traffic for different destination hosts are generated, each time with hundreds of flows. For DCQCN, the incast traffic quickly fills up ToR-Down ports' buffer, as well as core switches' buffer. When the incast occurs 12 times or more, the buffer occupancy on ToR-Up ports starts to increase, which indicates that the PFC pause frame storm happens. In Floodgate, the buffer of Core switches and ToR-Down ports are stable because the ToR-Up ports are the first hop gate-keepers of incast traffic. As a side-effect, the ToR-Up ports buffer occupancy increases at a rate proportional to the number of incast times when incast traffic arrives continuously. Nevertheless, it is notable that Floodgate can handle dozens of times of successive incast well. With *per-dst PAUSE*, the buffer occupancy is extremely small. Source hosts are paused by source ToRs. Therefore the in-network traffic can be reduced significantly.

There is a trade-off between deployability and robustness. Floodgate (per-dst PAUSE) is more robust in successive incast scenarios, but it requires coordination of source hosts. Floodgate does not require the help of source hosts, but it is limited to handle successive dozens of times of incast well. Lessons we learned are that incast can not be handled by congestion control protocols alone, therefore

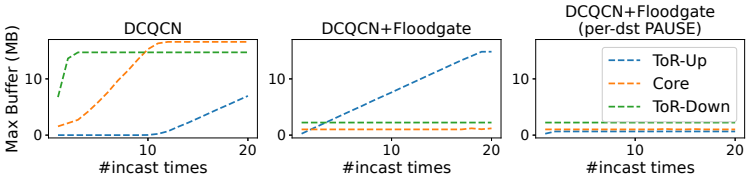
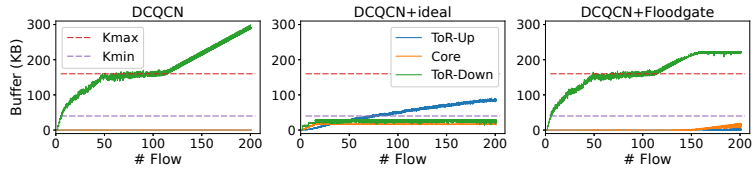
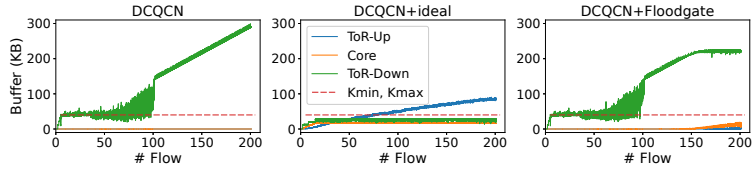


Figure 15: Comparison of Floodgate and Floodgate (per-dst PAUSE) when incast occurs multiple times successively.

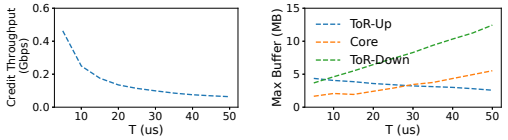


(a) ECN-marking threshold $K_{min} = 40KB$, $K_{max} = 160KB$

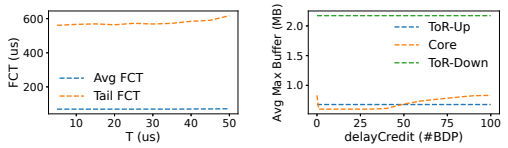


(b) ECN-marking threshold $K_{min} = 40KB$, $K_{max} = 40KB$

Figure 16: Realtime buffer occupancy under different ECN-marking thresholds. The x-stick i denotes the arrival of the i-th flow.

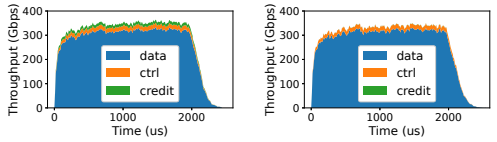


(a) Overhead under different T. (b) Buffer under different T.



(c) FCT under different T. (d) DelayCredit $thre_{credit}$

Figure 17: Parameter selection of credit timer T and delayCredit threshold $thre_{credit}$.



(a) Ideal (b) Floodgate

Figure 18: Stacking diagram of real time bandwidth occupation of data packets, credit packets and control packets (*i.e.*, ACKs or CNP).

we propose Floodgate. Furthermore, the fact is that to address incast problem, it is not only the responsibility of in-network switches nor end-point hosts, but it requires the co-design of both switches and hosts.

6.4 Compatible with CC

To dig into the convergence of Floodgate with different ECN-marking threshold settings, we conduct simulations where flows transmitting to the same receiver arrive periodically. The average interval between two consecutive flows is 7.7ms, which is long enough for convergence of congestion control. The buffer occupancy variation is shown in Figure 16. Two observations are found. First, the buffer occupancy of Floodgate (ideal) is insensitive to the ECN-marking threshold. Second, the buffer occupancy of DCQCN can not converge when flows continue arriving. For DCQCN, the buffer occupancy of ToR-Down has an inflection point, *i.e.*, the y-axis is K_{max}, and the corresponding x-axis is $max\{BW_{host}/Rate_{min}, K_{max}/MTU\}$. After that, because there is at least one in-flight packet for a flow, the buffer continues increasing along with the number of flows. While for Floodgate, the buffer occupancy of ToR-Down can converge to proportional to the initial sending window and topology scale.

6.5 Parameter Selection

In this section, Floodgate's performance under different parameter selections is evaluated. Figure 17 depicts the results.

Credit Timer T. Figure 17(a), (b) and (c) shows variation of network overhead, buffer occupancy, and FCT across different values of T. A larger T makes it less costly for switches to generate credit, and weakens the role Floodgate plays in reducing buffer occupancy, and vice versa. As shown in Figure 17(a), the network throughput used by credit packets decreases when T increases. To avoid bandwidth waste, the per-dst sending window is initialized to be proportional to T. The maximum buffer occupancy of ToR-Up ports decreases when T increases, as shown in Figure 17(b). For that source ToR can transmit more incast traffic to its downstream switches when the sending window increases. The traffic aggregation points, *i.e.*, core and ToR-Down ports, absorb data at a higher rate when T increases, resulting in a larger buffer occupancy. In addition, a larger T prolongs the average FCT and tail latency for that it controls the incast traffic less efficiently. There is a trade-off between performance and network overhead. In practice, T is set to 10μs.

DelayCredit $thre_{credit}$. Figure 17(d) shows the buffer occupancy variation with different $thre_{credit}$ values. $thre_{credit}$ determines how conservatively credits are sent back. When $thre_{credit}$ ranges from 1 to 38, Floodgate achieves the lowest buffer occupancy on core switches. At the same time, buffer occupancy on ToR-Up ports and ToR-Down ports remains almost unchanged. It indicates that Floodgate is robust with different values of $thre_{credit}$. *delayCredit* mechanism is much more useful in extreme scenarios where the number of ToR scales up (§ 6.2), because it helps to control the buffer occupancy on core switches. Generally, $thre_{credit}$ is set to 10 BDP.

7 HARDWARE FEASIBILITY

In this section, we dig into the hardware feasibility of Floodgate. We discuss the hardware requirements of Floodgate on both NICs (§ 7.1) and switches. A Floodgate pipeline in P4-based programmable switch ASIC is shown in Figure 19, consisting of ingress pipeline, traffic manager (TM) and egress pipeline, respectively (§ 7.2). For functions (*i.e.*, VOQ schedulers) that can not be supported by today's switch architecture, we give our envision on how to

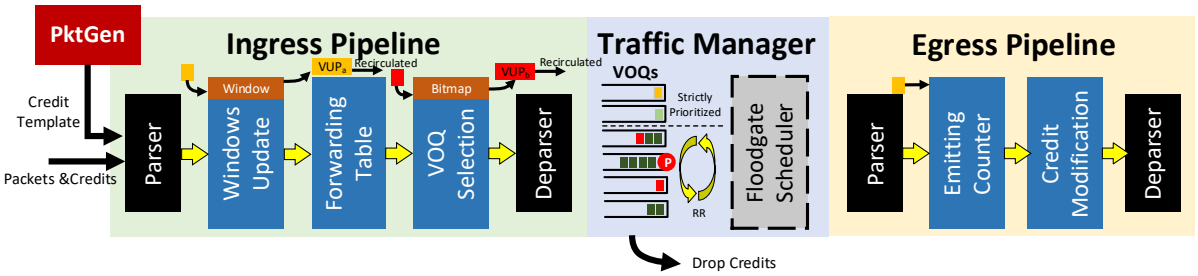


Figure 19: Pipeline of Floodgate switch.

implement it in future programmable switches (§ 7.3). At last, we analysis the memory and network overhead of Floodgate (§ 7.4).

7.1 Requirements on NICs

There are two versions of Floodgate, *i.e.*, ideal (§ 3.2) and practical design (§ 4). Floodgate does not require specific modifications on host NICs. Floodgate (ideal) leverages *dstPause* mechanism to achieve better performance (§ 4.3). It requests per-dst queues on hosts, which can be implemented by leveraging programmable smart NICs in RDMA networks [37]. It can also be supported in Ethernet networks, where per-dst queues can be implemented by modifying the QDisc kernel space [35].

7.2 Floodgate Pipeline

Ingress pipeline. *The ingress pipeline is responsible for maintaining the global per-dst sending window, packet forwarding, and VOQ selection.* When a packet arrives at the ingress, the parser parses the packet header. The window update module is leveraged to update the per-dst sending window. If a credit packet is received, increasing the corresponding sending window. And Floodgate replies on recirculated packets, called VOQ status updating packets (VUP), carrying back the number of packets dequeued from VOQs to increase the sending window. If the sending window is inadequate to forward a data packet (or the corresponding destination is already allocated with a VOQ), the metadata of the data packet is tagged with *no_win*. The forwarding table is used to calculate the egress port according to IP tuples in packets' header. Data packets without *no_win* tag are directly pushed into a default VOQ queue. VOQ selection module assigns data packets tagged with *no_win* to the left VOQs. If a data packet whose destination is already allocated with a VOQ, assigns it to the corresponding VOQ directly. Otherwise, it is allocated with an empty VOQ if possible, according to the VOQ bitmap. A bitmap of VOQs maintains the status of VOQ usage. In Tofino, only the egress pipeline can get the queuing status directly. Hence, to update the VOQ bitmap, VUP is leveraged to get the VOQ status. In addition, Tofino2 has already supported that the ingress pipeline stage subscribes to the status of specified queues [28].

Traffic Manager (TM). *TM contains the logic to manage packet buffer and schedule egress queues/VOQs.* When a data packet tagged with *no_win* arrives at the TM, it is pushed into the assigned VOQ. To update the bitmap in the ingress pipeline, VUP packets check the VOQ status and carry the information back recirculately. Floodgate switch maintains VOQ schedulers for each port. The queue reserved for non-incast traffic (*i.e.*, packets without the *no_win* tag) is strictly prioritized over the left VOQs, enabling non-incast traffic to be transmitted as soon as possible. And the scheduler inquiries the left VOQs based on a Round-Robin manner. Only VOQs with adequate sending windows can transmit data packets (see more details in § 7.3).

Egress pipeline. *The egress pipeline is responsible for constructing the credit packets, filling them with the number of credits.* Instead of leveraging explicit timers, credit packets are generated by the internal packet generator periodically, according to the preset interval. Tofino supports internal packet generator to inject 100 Gbps traffic into one Ethernet port [51]. Hence, we set the credit packets generation interval to a small value (*i.e.*,1μs). When a credit is generated, it is forwarded to an egress pipeline. The emitting counter table maintains the credits needed to be returned for each destination, *i.e.*, the number of packets dequeued from VOQs, and the timestamp recording the last sent time of credit packets. When a data packet is dequeued from the VOQ, the corresponding emitting counter is increased by one. When a credit packet arrives at the egress pipeline, checking the timestamp. If the passed time reaches a threshold T (*i.e.*, 10μs) and the emitting counter is not zero, the credit modification module modifies its credit field according to the emitting counter and resets it to zero. And the timestamp is updated. Otherwise, the credit packet is dropped. When a VUP arrives, it carries back the corresponding emitting counter back to the ingress pipeline to update the sending window.

7.3 Required Features for Future Architecture

To implement Floodgate, a scheduler should be applied to schedule the transmission of packets in VOQs according to the left sending window. Only packets whose destination has adequate sending windows can be forwarded.

However, to our knowledge, packet schedulers in current programmable switches can only support relatively simple scheduling algorithms, *e.g.*, Deficit Weighted Round-Robin (DWRR) [46]. The good news is that the programmable ability of the scheduler has raised great attention and has been investigated [48]. According to the discussions in [45], implementing a programmable scheduler with pausing features is relatively reasonable and straightforward. Because doing so does not change the order of temporal or spatial complexity of existing TM implementations. Moreover, PFC already requires a similar queue pausing/resuming capability triggered by specific protocol messages in the data plane.

In future programmable hardware, Floodgate's transmission scheduler can be implemented in the following way. When a packet is received, if the remaining window is not enough, a *no_win flag* is set in the packet's metadata, then the traffic manager will pause the transmission of the corresponding VOQ. VUP is used to notify the traffic manager when sending window is inadequate in case there

are no following data packets. And when a credit is received, the traffic manager resumes the corresponding VOQ. We are currently in cooperation with a leading switch vendor to add this feature in its next-generation switch.

7.4 Resource Overhead

Memory overhead. The memory overhead comes from the runtime status that Floodgate should maintain. Floodgate's switch needs to maintain sending windows for active destination (destination with full window size can be cleared periodically). In the worst case, the maximum number of sending windows scales with the number of hosts in the network. For a datacenter network consisting of 100,000 servers, entries of sending window equal 100,000 in the worst case, which consumes less than 10% of the switch's dedicated stateful memory according to discussions in BFC (Section 3.3.1) [19]. Besides, given that incast traffic is isolated from non-incast traffic, small non-incast flows can be finished quickly. Therefore, the number of active destinations can be modest. In addition, destination IPs can be hashed into hash tables to store the value of sending windows, saving memory at the cost of sacrificing precision.

Network overhead. To evaluate Floodgate's network overhead, real-time bandwidth occupation is monitored on switches' egress ports. The bandwidth occupation of three different kinds of packets is shown in Figure 18. Control packets, *i.e.*, ACKs or CNP sent by receiver hosts, saturate 4.5% of bandwidth in both approaches, the same as in DCQCN. For Floodgate, credit packets saturate 0.175% of bandwidth, and that value is 3.0% in ideal design. It suggests that the network overhead induced by Floodgate is negligible.

8 DISCUSSION

Compatible with different congestion control. In Floodgate, data packets recognized as incast could be pushed into VOQs. For congestion control protocols leveraging ECN or egress queue length as the congestion signals, *i.e.*, DCTCP/DCQCN/HPCC, Floodgate's switch should maintain a counter to perform the same function. When a packet arrives at the ingress port, it increases the counter after determining the forwarding egress port. And when a packet is forwarded to the next hop, the counter is decreased by one. In Floodgate, buffer occupancy on ToR could be larger than before. HPCC is sensitive to queue length, hence, the flow rate of non-incast flows could be reduced aggressively. Therefore, to cooperate with HPCC, the queue length carried in non-incast packets is set to the egress queue length, while the queue length carried in incast packets (*i.e.*, packets queuing at VOQs) is set to the sum of VOQs.

Compared with BFC. BFC is a congestion control architecture that leverages per-hop per-flow flow control, aiming at improving both latency for short flows and throughput for long flows. BFC acts based on a pause/resume manner, at a per-queue granularity. BFC assigns flows to a limited number of queues, *i.e.*, 32/128 per port in Tofino2. When a packet arrives, check the corresponding egress queue length. Once it just exceeds a given threshold, a pause frame is sent to the upstream port to pause the transmission of the corresponding queue. And the upstream queue is resumed once all its packets that exceeded the pause threshold have been transmitted. BFC has already been implemented on Tofino2, a state-of-the-art P4-based programmable switch ASIC.

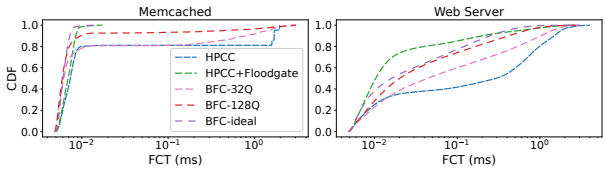


Figure 20: Compared with BFC under incastmix scenarios.

BFC can not avoid HOL blocking caused by incast flows totally. Figure 20 compares the performance of BFC and Floodgate. We evaluate three versions of BFC. BFC-ideal uses infinite queues, meanwhile, each flow is allocated with a dedicated queue, and flow identifier (FID) is assigned to flow-id (*i.e.*, no hash collision occurs). For BFC-32Q/128Q, different flows could share the same queue. It is obvious that when queues are used up, non-incast flows could share the same queue with incast flows. Moreover, we find that even when queues are adequate, non-incast flows and incast flows could also share the same queue (see Appendix B for details). Due to the fact that the pausing mechanism of BFC works on a per-queue granularity, unrelated flows sharing the same upstream queue will get paused even though they are not going through the congested port (*i.e.*, congestion spreading occurs). HOL blocking could happen, especially when non-incast flows share the same queue with incast flows. This is the reason why BFC-32Q/128Q performs poorer than Floodgate. BFC-ideal performs better than Floodgate under Memcached, in that INT used by HPCC wastes bandwidth. While under Web server, Floodgate performs better than BFC-ideal. Because Floodgate recognizes incast flows and tames the transmission of incast flows quickly, thus avoiding bandwidth waste at the last-hop.

Additional comparison. We compare Floodgate with NDP, and a derive of Floodgate named PFC w/ tag in Appendix B to dig into more characteristics of Floodgate.

9 CONCLUSION

This paper analyzes the consequence of incast, and proposes Floodgate, a switch-based hop-by-hop flow control, which handles incast by following steps: (i) recognize incast flows quickly and accurately; (ii) tame the transmission of incast flows; (iii) isolate incast flows from non-incast flows. Floodgate reduces the buffer occupancy significantly, thus reducing packet loss/PFC. Therefore, the performance of non-incast flows is greatly improved while the performance of incast flows is not compromised. With the rapid growth of switches' programmability, co-design of congestion control with switch-based hop-by-hop flow control is promising.

ACKNOWLEDGMENTS

We thank our shepherd Mina Tahmasbi Arashloo and the anonymous CoNEXT reviewers for their valuable feedback. This research is supported by the Key-Area Research and Development Program of Guangdong Province 2020B0101390001, the National Natural Science Foundation of China under Grant Numbers 61772265 and 62072228, the Fundamental Research Funds for the Central Universities, the Collaborative Innovation Center of Novel Software Technology and Industrialization, and the Jiangsu Innovation and Entrepreneurship (Shuangchuang) Program.

REFERENCES

[1] 2006. Apache Hadoop. http://hadoop.apache.org. (2006).
[2] 2010. Apache Spark. http://spark.apache.org/. (2010).
[3] Alibaba. 2019. HPCC simulator. https://github.com/alibaba-edu/High-Precision-Congestion-Control. (2019).
[4] Mohammad Alizadeh, Albert Greenberg, David A Maltz, Jitendra Padhye, Parveen Patel, Balaji Prabhakar, Sudipta Sengupta, and Murari Sridharan. 2011. Data center TCP (DCTCP). In *ACM SIGCOMM*.
[5] Infiniband Trade Association. 2014. Supplement to InfiniBand architecture specification volume 1 release 1.2.2 annex A17: RoCEv2 (IP routable RoCE). (2014).
[6] Wei Bai, Kai Chen, Haitao Wu, Wuwei Lan, and Yangming Zhao. 2014. PAC: Taming TCP Incast Congestion Using Proactive ACK Control. In *2014 IEEE 22nd International Conference on Network Protocols*.
[7] Ran Ben Basat, Sivaramakrishnan Ramanathan, Yuliang Li, Gianni Antichi, Minian Yu, and Michael Mitzenmacher. 2020. PINT: Probabilistic In-Band Network Telemetry. In *ACM SIGCOMM*.
[8] Berkeley. 2018. Berkeley Extensible Software Switch. https://github.com/NetSys/bess. (2018).
[9] Pat Bosshart, Glen Gibb, Hun-Seok Kim, George Varghese, Nick McKeown, Martin Izzard, Fernando Mujica, and Mark Horowitz. 2013. Forwarding Metamorphosis: Fast Programmable Match-Action Processing in Hardware for SDN. *SIGCOMM*.
[10] Broadcom. 2021. BCM88800 Traffic Management Architecture. https://docs.broadcom.com/doc/88800-DG1-PUB. (2021).
[11] Inho Cho, Keon Jang, and Dongsu Han. 2017. Credit-scheduled delay-bounded congestion control for datacenters. In *ACM SIGCOMM*.
[12] Cisco. 2010. Cisco Nexus 5548P Switch Architecture. https://www.cisco.com/c/en/us/products/collateral/switches/nexus-5548p-switch/white_paper_c11-622479.html. (2010).
[13] Austin Donnelly, Greg O'Shea, Ant Rowstron, and Paolo Costa. 2012. Camdoop: Exploiting In-network Aggregation for Big Data Applications. In *NSDI*.
[14] Linux Foundation. 2015. Data Plane Development Kit (DPDK). http://www.dpdk.org. (2015).
[15] Peter X Gao, Akshay Narayan, Sagar Karandikar, Joao Carreira, Sangjin Han, Rachit Agarwal, Sylvia Ratnasamy, and Scott Shenker. 2016. Network requirements for resource disaggregation. In *USENIX OSDI*.
[16] Peter X Gao, Akshay Narayan, Gautam Kumar, Rachit Agarwal, Sylvia Ratnasamy, and Scott Shenker. 2015. pHost: Distributed near-optimal datacenter transport over commodity network fabric. In *ACM CoNEXT*.
[17] Yixiao Gao, Qiang Li, Lingbo Tang, Yongqing Xi, Pengcheng Zhang, Wenwen Peng, Bo Li, Yao hui Wu, Shaozong Liu, Lei Yan, Fei Feng, Yan Zhuang, Fan Liu, Pan Liu, Xingkui Liu, Zhongjie Wu, Junping Wu, Zheng Cao, Chen Tian, Jinbo Wu, Jiaji Zhu, Haiyong Wang, Dennis Cai, and Jiesheng Wu. 2021. When Cloud Storage Meets RDMA. In *NSDI*.
[18] Yixiao Gao, Yuchen Yang, Chen Tian, Jiaqi Zheng, Bing Mao, and Guihai Chen. 2018. DCQCN+: Taming Large-Scale Incast Congestion in RDMA over Ethernet Networks. *2018 IEEE 26th International Conference on Network Protocols (ICNP)* (2018), 110–120.
[19] Prateesh Goyal, Preey Shah, Naveen Kr. Sharma, Mohammad Alizadeh, and Thomas E. Anderson. 2022. Backpressure Flow Control. In *NSDI*.
[20] The P4.org Architecture Working Group. 2021. $P4_{16}$ Portable Switch Architecture (PSA). https://p4lang.github.io/p4-spec/docs/PSA.pdf. (2021).
[21] Chuanxiong Guo, Haitao Wu, Zhong Deng, Gaurav Soni, Jianxi Ye, Jitu Padhye, and Marina Lipshteyn. 2016. RDMA over commodity Ethernet at scale. In *ACM SIGCOMM*.
[22] Y. Guo. 2001. Hydrologic Design of Urban Flood Control Detention Ponds. *Journal of Hydrologic Engineering* 6 (2001), 472–479.
[23] Mark Handley, Costin Raiciu, Alexandru Agache, Andrei Voinescu, Andrew W Moore, Gianni Antichi, and Marcin Wójcik. 2017. Re-architecting datacenter networks and stacks for low latency and high performance. In *ACM SIGCOMM*.
[24] Shuihai Hu, Wei Bai, Baochen Qiao, Kai Chen, and Kun Tan. 2018. Augmenting Proactive Congestion Control with Aeolus. In *Proceedings of the 2nd Asia-Pacific Workshop on Networking*. ACM, 22–28.
[25] Shuihai Hu, Wei Bai, Gaoxiong Zeng, Zilong Wang, Baochen Qiao, Kai Chen, Kun Tan, and Yi Wang. 2020. Aeolus: A Building Block for Proactive Transport in Datacenters. In *ACM SIGCOMM*. ACM.
[26] Huawei. 2021. CloudEngine 12800 Series Data Center Core Switches. https://docs.broadcom.com/doc/88800-DG1-PUB. (2021).
[27] IEEE. 2011. 802.11Qbb. Priority based flow control. https://1.ieee802.org/dcb/802-1qbb/. (2011).
[28] Intel. 2020. Intel Tofino2 – A 12.9Tbps P4-Programmable Ethernet Switch. https://ieeexplore.ieee.org/document/9220636. (2020).
[29] Juniper. 2017. Understanding CoS Virtual Output Queues (VOQs) on QFX10000 Switches. https://www.juniper.net/documentation/en_US/junos/topics/concept/cos-qfx-series-voq-understanding.html. (2017).
[30] M. Karol, S. J. Golestani, and D. Lee. 2003. Prevention of deadlocks and livelocks in lossless backpressured packet networks. *IEEE/ACM Transactions on Networking* (2003).
[31] Gautam Kumar, Nandita Dukkipati, Keon Jang, Hassan Wassel, Xian Wu, Behnam Montazeri, Yaogong Wang, Kevin Springborn, Christopher Alfeld, Michael Ryan, David Wetherall, and Amin Vahdat. 2020. Swift: Delay is Simple and Effective for Congestion Control in the Datacenter. In *ACM SIGCOMM*.
[32] Yuliang Li, Rui Miao, Hongqiang Harry Liu, Yan Zhuang, Fei Feng, Lingbo Tang, Zheng Cao, Ming Zhang, Frank Kelly, Mohammad Alizadeh, et al. 2019. HPCC: High precision congestion control. In *ACM SIGCOMM*.
[33] Hwijoon Lim, Wei Bai, Yibo Zhu, Youngmok Jung, and Dongsu Han. 2021. Towards Timeout-Less Transport in Commodity Datacenter Networks. In *Proceedings of the Sixteenth European Conference on Computer Systems*.
[34] Kexin Liu, Guihai Chen, Wanchun Dou, Yanan Jiang, Huaping Zhou, Jingjie Jiang, Fan Zhang, Gong Zhang, Bingchuan Tian, Chen Tian, Bo Li, Qingyue Wang, Jiaqi Zheng, Jiajun Sun, Yixiao Gao, and Wei Wang. 2020. Exploring Token-Oriented In-Network Prioritization in Datacenter Networks. *IEEE Transactions on Parallel and Distributed Systems* (2020).
[35] Shiyu Liu, Ahmad Ghalayini, M. Alizadeh, B. Prabhakar, M. Rosenblum, and Anirudh Sivaraman. 2021. Breaking the Transience-Equilibrium Nexus: A New Approach to Datacenter Packet Transport. In *NSDI*.
[36] Vincent Liu, Daniel Halperin, Arvind Krishnamurthy, and Thomas Anderson. 2013. F10: A Fault-Tolerant Engineered Network. In *NSDI 13*. USENIX Association.
[37] Jaehong Min, Ming Liu, Tapan Chugh, Chenxingyu Zhao, Andrew Wei, In-Hwan Doh, and Arvind Krishnamurthy. 2021. Gimbal: enabling multi-tenant storage disaggregation on SmartNIC JBOFs. *Proceedings of the 2021 ACM SIGCOMM 2021 Conference* (2021).
[38] Radhika Mittal, Nandita Dukkipati, Emily Blem, Hassan Wassel, Monia Ghobadi, Amin Vahdat, Yaogong Wang, David Wetherall, David Zats, et al. 2015. TIMELY: RTT-based congestion control for the datacenter. In *ACM SIGCOMM*.
[39] Behnam Montazeri, Yilong Li, Mohammad Alizadeh, and John Ousterhout. 2018. Homa: A receiver-driven low-latency transport protocol using network priorities. In *ACM SIGCOMM*.
[40] David Nagle, Denis Serenyi, and Abbie Matthews. 2004. The Panasas ActiveScale Storage Cluster: Delivering Scalable High Bandwidth Storage. In *Supercomputing*. ACM.
[41] NASA NJU. 2021. Floodgate simulator. https://github.com/NASA-NJU/Floodgate-NS3. (2021).
[42] W. W. Peterson and D. T. Brown. 1961. Cyclic Codes for Error Detection. *Proceedings of the IRE* (1961).
[43] Arjun Roy, Hongyi Zeng, Jasmeet Bagga, George Porter, and Alex C. Snoeren. 2015. Inside the Social Network's (Datacenter) Network. In *Proceedings of the 2014 ACM conference on SIGCOMM*.
[44] Yizhou Shan, Yutong Huang, Yilun Chen, and Yiying Zhang. 2018. Legoos: A disseminated, distributed OS for hardware resource disaggregation. In *USENIX OSDI*.
[45] Naveen Kr. Sharma, Chenxingyu Zhao, Ming Liu, Pravein G. Kannan, Changhoon Kim, A. Krishnamurthy, and Anirudh Sivaraman. 2020. Programmable Calendar Queues for High-speed Packet Scheduling. In *NSDI*.
[46] M. Shreedhar and G. Varghese. 1996. Efficient fair queueing using deficit round-robin. *IEEE/ACM Trans. Netw.* 4 (1996), 375–385.
[47] Vishal Shrivastav, Asaf Valadarsky, Hitesh Ballani, Paolo Costa, Ki Suh Lee, Han Wang, Rachit Agarwal, and Hakim Weatherspoon. 2019. Shoal: A network architecture for disaggregated racks. In *USENIX NSDI*.
[48] Anirudh Sivaraman, Suvinay Subramanian, Anurag Agrawal, S. Chole, Shang-Tse Chuang, T. Edsall, M. Alizadeh, S. Katti, N. McKeown, and H. Balakrishnan. 2015. Towards Programmable Packet Scheduling. *Proceedings of the 14th ACM Workshop on Hot Topics in Networks* (2015).
[49] Sage A Weil, Scott A Brandt, Ethan L Miller, Darrell DE Long, and Carlos Maltzahn. 2006. Ceph: A scalable, high-performance distributed file system. In *USENIX OSDI*. ACM.
[50] Wikipedia. 2020. Floodgate. https://wikipedia.org/wiki/Floodgate. (2020).
[51] Dingming Wu, Ang Chen, T. S. Eugene Ng, Guohui Wang, and Haiyong Wang. 2019. Accelerated Service Chaining on a Single Switch ASIC. *Proceedings of the 18th ACM Workshop on Hot Topics in Networks* (2019).
[52] Haitao Wu, Z. Feng, C. Guo, and Y. Zhang. 2013. ICTCP: incast congestion control for TCP in data-center networks. *IEEE/ACM Trans. Netw.* 21 (2013), 345–358.
[53] Haitao Wu, Jiabo Ju, Guohan Lu, Chuanxiong Guo, Yongqiang Xiong, and Yongguang Zhang. 2012. Tuning ECN for data center networks. In *ACM CoNEXT*.
[54] Jiao Zhang, Fengyuan Ren, Li Tang, and Chuang Lin. 2013. Taming TCP incast throughput collapse in data center networks. In *2013 21st IEEE International Conference on Network Protocols (ICNP)*.
[55] Yibo Zhu, Haggai Eran, Daniel Firestone, Chuanxiong Guo, Marina Lipshteyn, Yehonatan Liron, Jitendra Padhye, Shachar Raindel, Mohamad Haj Yahia, and Ming Zhang. 2015. Congestion control for large-scale RDMA deployments. In *ACM SIGCOMM*.

APPENDIX

A SUPPLEMENTAL RESULTS

A.1 Incast Flows's Performance Under Incastmix Scenarios

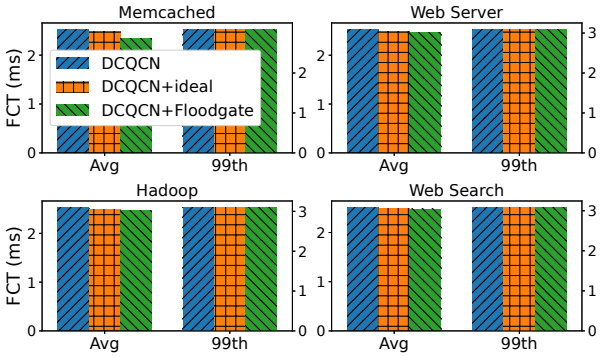


Figure 21: FCT of incast flows under incastmix scenarios.

Figure 21 corresponds to the simulation described in § 6.1. Floodgate does not downgrade the performance of incast flows, which indicates that the bandwidth of incast flows is utilized. Furthermore, a little improvement is observed. This is because Floodgate tames the transmission of incast flows, avoiding a large queuing delay at the last hop. For ideal design, the incast FCT is slightly increased. For that it is strict for incast flows, and gives Poisson flows a more significant improvement than Floodgate (as shown in Figure 8).

A.2 Pure Poisson Scenarios

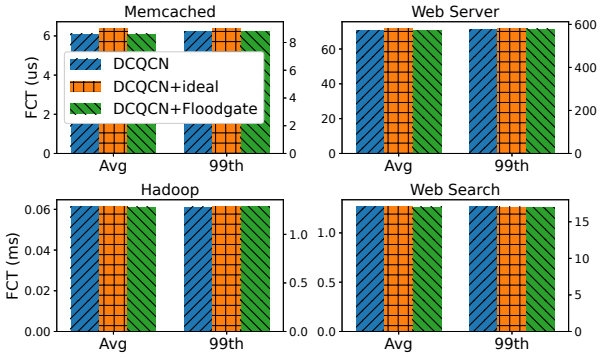
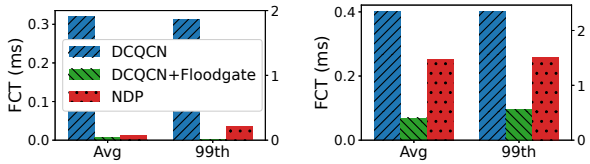


Figure 22: FCT under poisson scenarios.

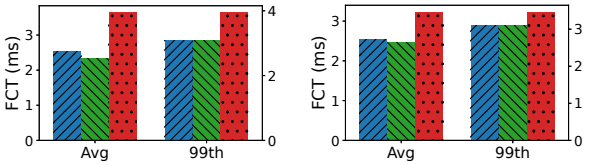
Figure 22 shows the performance under pure Poisson arrival flow scenarios. For DCQCN+Floodgate, no non-incast flows are wrongly recognized as incast, and the overhead of Floodgate is negligible. Therefore the performance of DCQCN+Floodgate is almost the same as DCQCN. The performance of DCQCN+ideal is slightly worse than DCQCN because of the overhead of credit (§ 7.4).

B ADDITIONAL DISCUSSIONS

Compared with NDP. Figure 23 shows the performance of NDP, DCQCN and DCQCN + Floodgate. MTU is set to 1.5 KB. For non-incast flows, benefiting from a relatively small buffer occupancy, NDP performs better than DCQCN. However, when compared with DCQCN + Floodgate, a significant FCT performance downgrade is

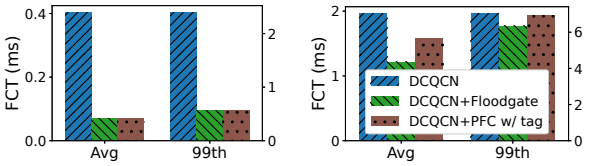


(a) Non-incast flows under Memcached. (b) Non-incast flows under WebServer.



(c) Incast flows under Memcached. (d) Incast flows under WebServer.

Figure 23: Compared with NDP under incastmix scenarios.



(a) Under non-blocking topology. (b) Under oversubscribed topology (4:1 oversubscription ratio).

Figure 24: Compared with PFC w/ tag.

observed. For that NDP does not identify incast flows from non-incast flows, letting all flows suffer from packet trimming and retransmission. When incast flows already have depleted the queue length to the cut-payload threshold, following non-incast flows can also encounter packet trimming. It costs data packets at least one RTT for retransmission. The performance of non-incast flows could get downgraded. In addition, NDP prolongs the FCT performance of incast flows, as shown in Figure 23(c) and (d). This is because packet headers (CP) uses much bandwidth, *i.e.*, 30.5% and 28% of bottleneck bandwidth is used under Memcached and WebServer workloads, respectively. Hence, the network goodput is downgraded.

BFC can not mitigate HOL blocking. By digging into the simulation results of BFC, we observed that non-incast flows and incast flows could share the same queue when queues are adequate. There are briefly three reasons.

- (i) When an incast flow is dequeued, the queue it used could be newly assigned to the following non-incast flows. Since a switch does not know the current queue assignment of a flow at the upstream, it uses the *upstreamQ* conveyed by the incast flow to pause a queue. Innocent non-incast flows could be paused only because they use the same queue that incast flows used before.
- (ii) To control the transmission of incast flows more efficiently, BFC leverages sticky queue assignment. A sticky queue denotes that a flow is always assigned to the same queue for a period of time (*i.e.*, sticky threshold). When incast flows arrive periodically, the following incast flows can be paused before forwarding to the downstream switch. However, it has a side-effect: when the sticky queue used by incast flows is empty, non-incast flows could be pushed into it. Therefore, following incast flow could share the same queue with non-incast flows.

- (iii) Hash functions are used to keep track of the queue assignment. Hash collision could occur. We observed that at most eight flow IDs could be hashed into the same FIDs simultaneously.

PFC w/ tag discussion. A derive of Floodgate , *i.e.*, called PFC w/ tag, acts as below. When the egress queue length of the last-hop ToR switch exceeds a threshold, a pause frame tagged with the destination experiencing incast is sent to the upstream switch. PFC w/ tag is different from PFC in that it states what destination to pause rather than the entire traffic between two switches, and it works on the egress queue. When the upstream switch receives a pause frame, upcoming packets whose destination is paused are pushed into a dedicated VOQ. Likewise, if the VOQ queue length of the switch exceeds the PFC threshold, pause frames can be sent to its upstream further. In addition, to avoid unnecessary transmission of resume frames, the switch should record the paused upstream entities for each destination.

PFC w/ tag and Floodgate both detect per-dst incast, but they act in a different way. PFC w/ tag detects incast and then generates pause frames according to the length of the egress queue instead of keeping track of the in-flight packets as Floodgate does. PFC w/ tag is a reactive flow control while Floodgate is a proactive flow control. Figure 24 compares the performance of PFC w/ tag and Floodgate. The performance of Floodgate is comparable with PFC w/ tag in non-blocking topology. In addition, the number of VOQs used by PFC w/ tag is larger than Floodgate by an order of magnitude, indicating that PFC w/ tag does not recognize incast flows accurately. For that PFC w/ tag has a longer control loop; a smaller threshold should be set to detect incast quickly. Under blocking topology, Floodgate achieves better performance than PFC w/ tag. Since that for PFC w/ tag, when the first-hop ToR switch becomes the congested point, the last-hop detection of incast is not timely enough, *i.e.*, only when incast traffic arrives at the last-hop ToR switch will a pause frame start to be transmitted. Non-incast flows can already have experienced HOL blocking by incast flows before PFC w/ tag taking effects. While for Floodgate, incast can be detected at the first-hop switch. Likewise, along with the number of ToR switches increasing, the incast on core switches scales up, which could downgrade the performance of PFC w/ tag.