Backdraft: a Lossless Virtual Switch that Prevents the Slow Receiver Problem

Alireza Sanaee, *Queen Mary University of London;* Farbod Shahinfar, *Sharif University of Technology;* Gianni Antichi, *Queen Mary University of London;* Brent E. Stephens, *University of Utah*

https://www.usenix.org/conference/nsdi22/presentation/sanaee

This paper is included in the Proceedings of the 19th USENIX Symposium on Networked Systems Design and Implementation.

April 4–6, 2022 • Renton, WA, USA

978-1-939133-27-4

Open access to the Proceedings of the 19th USENIX Symposium on Networked Systems Design and Implementation is sponsored by

Backdraft: a Lossless Virtual Switch that Prevents the Slow Receiver Problem

Alireza Sanaee[†], Farbod Shahinfar[*], Gianni Antichi[†], Brent E. Stephens[‡]
[†]*Queen Mary University of London,* [*]*Sharif University of Technology,* [‡]*University of Utah*

Abstract

Virtual switches, used for end-host networking, drop packets when the receiving application is not fast enough to consume them. This is called the slow receiver problem, and it is important because packet loss hurts tail communication latency and wastes CPU cycles, resulting in application-level performance degradation. Further, solving this problem is challenging because application throughput is highly variable over short timescales as it depends on workload, memory contention, and OS thread scheduling.

This paper presents Backdraft, a new *lossless virtual switch* that addresses the slow receiver problem by combining three new components: (1) Dynamic Per-Flow Queuing (DPFQ) to prevent HOL blocking and provide on-demand memory usage; (2) Doorbell queues to reduce CPU overheads; (3) A new overlay network to avoid congestion spreading. We implemented Backdraft on top of BESS and conducted experiments with real applications on a 100 Gbps cluster with both DCTCP and Homa, a state-of-the-art congestion control scheme. We show that an application with Backdraft can achieve up to 20x lower tail latency at the 99th percentile.

1 Introduction

Virtual switches (vswitches) play an important role in today's data center networks operation [30, 33, 38, 68]. They are in charge of routing packets to one of the many competing microservices and applications running on a server that are communicating both locally and remotely [48, 66, 86]. They also provide isolation [61, 68, 87], enable load balancing [51], and perform packet encapsulation and decapsulation for secure virtual networking [30, 38, 39].

Virtual switches are fundamentally different from their physical counterpart. A physical switch has fixed port bandwidth, and its draining rate of output queues does not change over time. This is not the case for vswitches, as their draining rate of output queues depends on the ability of connected applications to consume packets. When packets arrive faster than an application can process, queues inside the vswitch fill up and overflow, leading to packet loss. This is called the *slow receiver problem* [21, 44, 60, 73], and it hurts tail network communication latency and wastes CPU cycles, impacting application-level performance [22, 27, 91, 96].

In this paper, we show that slow receivers can manifest at short timescales and cause packet loss even in the presence of state-of-the-art congestion controls such as Homa [72] (§2.1). Moreover, CPU cycles are wasted in handling dropped packets, and this further increases latency and the already high software overheads of current network stacks [21, 72, 73], inflating the problem. Although there are existing approaches to mitigate packet loss (*i.e.*, bandwidth reservation [13, 49, 50], backpressure [31, 43], credit-based hop-by-hop flow control [62], PicNIC [61]), they all have key limitations (§2.2). For example, because virtual ports bandwidth are variable over time, reservation schemes either lead to reduced network throughput or fail to prevent packet loss. Today's backpressure flow control solutions suffer from severe Head-of-Line (HOL) blocking and congestion spreading, leading to reduced throughput across the entire cluster [44, 88, 99] and unacceptable latency for some applications [16, 65]. PicNIC [61, 79], a state-of-the-art solution to provide predictable performance in a multi-tenant data center, incurs high CPU utilization and consequent throughput degradation and HOL blocking for flows sharing a Virtual Machine (VM).

To prevent packet loss from the slow receiver problem, this paper presents Backdraft, a new lossless vswitch. Backdraft prevents packet loss while (1) avoiding HOL blocking, (2) reducing the required CPU cycles, and (3) preventing congestion spreading in the network core (§3). Our main insight is that, unlike physical switches, vswitches have abundant memory that can be used to support a large number of queues.

Leveraging this property, Backdraft assigns a separate queue for every single flow, preventing HOL blocking. To ensure that per-flow queuing is not prohibitive in its memory overheads, we introduce an approach that dynamically reclaims queues from idle flows and resizes them to accommodate in-flight packets from bursty flows.

Also, Backdraft uses separate queues for doorbells (notifications) and packet data to reduce the CPU overhead induced by per-flow queueing that can impact the vswitch performance. In this approach, the vswitch has only to poll the doorbell queue to find where the new to be processed data is located. By keeping the number of doorbell queues low, it is possible to greatly reduce CPU overheads, enabling per-flow data queueing and scaling to 100 Gbps switching performance.

Finally, Backdraft uses an overlay network between communicating vswitches. When a queue inside the vswitch begins to fill because of a slow receiver, Backdraft preemptively sends an Overlay Pause Frame (OPF) to the upstream vswitch responsible for the congestion with pause time and the slow receiver's bandwidth. This is practical because vswitches have a large amount of memory that can be used to store in-flight packets generated by the sender before receiving the OPF notification. Indeed, even buffering a full RTT of packets in a 100 Gbps network, a worst case of 1ms RTT would only require 12.5 MB of space (1 Bandwidth-Delay-Product - BDP), and end-hosts have GBytes of memory.

We implemented Backdraft on top of the BESS vswitch [3, 45] (§4), and evaluated it using a cluster of servers on CloudLab [75] equipped with 10 and 100 Gbps NICs (§6). We experimented with both standard and state-of-the-art congestion controls: in the first case we used unmodified POSIX applications leveraging the TAS TCP acceleration service [56]; In the second, we used Homa [72] with its DPDK implementation. When we ran a distributed application that performs RPCs, Backdraft in conjuction with Homa could lower its tail latency by up to 20x at the 99^{th} percentile. With Memcached, instead, Backdraft could improve its goodput by up to 2.71x when compared to BESS. We also show that Backdraft does not suffer by HOL blocking and because of this can achieve 100 Gbps throughput in a cluster where a slow receiver is present. Finally, we demonstrate that Backdraft ensures high throughput with large number of queues. With 2K queues, throughput is 9x higher than BESS. This paper makes three contributions:

1. We make the case for building a lossless virtual switch by demonstrating the impact of slow receivers on packet loss and network performance using both DCTCP and Homa, a state-of-the-art congestion control algorithm.
2. We introduce Backdraft, a new lossless virtual switch that prevents the slow receiver problem and overcomes the drawbacks of state-of-the-art solutions: It (1) prevents packet loss, (2) removes HOL blocking, (3) increases throughput by eliminating wasted CPU cycles, and (4) avoids congestion spreading in the core network.
3. We implement and evaluate Backdraft on top of BESS using different congestion control mechanisms in a cluster of servers on CloudLab equipped with 10 Gbps and 100 Gbps NICs. We released our code under a flexible open-source license to enable reproducibility[1].

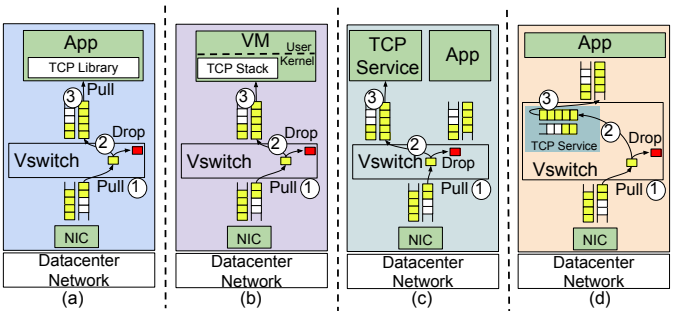


Figure 1: Various deployments of transport layer with respect to vswitches. (a) transport as a library. (b) transport as an OS service. (c) transport as a network function. (d) transport as a vswitch service.

2 Motivation

Virtual switches use shared memory queues to transmit and receive packets to and from connected end-points (Figure 1). Here, depending on the settings, the transport layer can be directly included into the application as a library (case a) [56], deployed in the kernel of a virtual machine (case b), used as a network service directly attached to the vswitch (case c) [59], or implemented in the vswitch (case d) [68]. Regardless, whenever the vswitch is ready to handle new data coming from the wire, it pulls a packet pointer from one of the NIC queues (point 1), performs processing and places it in the queue associated to the destination endpoint (point 2). Finally, the endpoint pulls the pointer and consumes the data (point 3). If this last step is not fast enough, the queue saturates and packets will be dropped at the vswitch. Notably, the discussed queue is not subjected to transport-level flow control mechanisms, so even if an endpoint has enough memory reserved for incoming packets (for example, TCP's receive window ensures there is space in the receive buffer), it is still possible for packets to arrive faster than the endpoint can process them and eventually get dropped. This issue has been acknowledged in the past, and it is called *the slow receiver problem* [21, 44].

2.1 The Slow Receiver Problem

There are many reasons for slow receivers, including allocation limitations [81], application-level limitations, load imbalance [19, 28, 32, 51, 52, 63, 71, 74], CPU performance variability [17, 25, 37, 42, 54, 66, 82, 97], and CPU/Memory contention [14, 35, 40, 41, 67].

To better understand this, we performed a number of tests on a 100 Gbps cluster (more information available in §6). First, we measured the achievable throughput of data-intensive (i.e., Nginx [8] and Memcached [7]) and network-only applications (iperf3 [6]) using an increasing number

[1] https://github.com/Lossless-Virtual-Switching/Backdraft

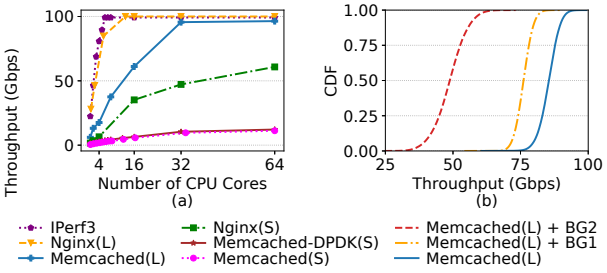


Figure 2: Maximum achieved throughput by Memcached, Nginx, and Iperf running with DPDK and Linux with large (L) and small (S) response sizes. (a) They require more than 6 cores to achieve 100 Gbps. (b) Memcached exhibits high throughput variability with and without the background workload. (BG1: on isolated cores. BG2: on shared cores)

of assigned processing cores. We also used different packet I/O frameworks (*e.g.*, standard Linux socket and DPDK) and different workloads. For Memcached, we used both small (200 B) and large values (4.8 KB) with sizes inspired by an analysis of caching at Twitter [92]. For Nginx, we served both small (4.8 KB) and large (1 MB) web pages.

Figure 2a shows the result of this experiment. We find that even iperf3, an application that only performs networking functionalities and no other specific processing, cannot hit 100 Gbps throughput with less than 6 cores. For other applications, even 64 cores might not be enough. Further, performance is highly dependent on the specific workload: Memcached using the Linux socket interface and serving 4.8 KB values with 32 cores achieves 16x higher bandwidth than the counterpart serving 200 B values. In contrast, Memcached can achieve 187 KRPS per core when serving 200 B items, while only 78 KRPS when serving 4.8 KB items.

Resource provisioning (OS scheduling) also plays a key role in application behavior [21, 73]. To better understand this, we run Memcached with 32 threads solely on bare-metal servers, where each thread resides on a separate logical core (the number of total logical cores is 64). Then, we use sysbench [58], which only exercises 32 logical cores, along with Memcached on the same machine. We evaluated both scenarios when Memcached and sysbench share CPU cores and when the two applications are isolated on different cores. Figure 2b shows that the Memcached server is unpredictable even *without* a background workload. When it is run with sysbench, its performance degrades by 12% or by 50% depending on the amount of contention. Moreover, the standard deviation of the throughput distribution increases by up to 1.71x.

Even worse, applications behavior can be highly variable and dependent on the workload [92]. We show this with experiments using Memcached and Nginx. To test the former, we used four clients generating a workload resembling the one experienced by Facebook [15]. For the latter, we used sixty single threaded clients requesting data from a copy of

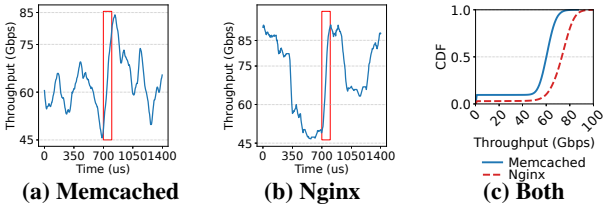


Figure 3: Throughput variability of Memcached (a) and Nginx (b) in a 1.4ms window. Throughput is highly variable over short timespans: *box is 100 μs*. In the box, we can see over 40 Gbps variability in less than 100 μs. (c) CDF of Memcached and Nginx throughput over the entire experiment.

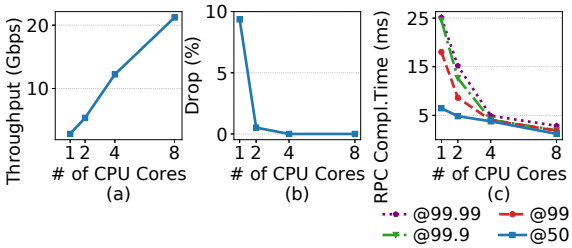


Figure 4: Throughput (a), packet loss (b) and RPC completion time (c) for a bidirectional RPC using Homa, the state-of-the-art transport protocol for data center networks. Packet loss can reach even 10% when only one core is assigned to the server application. RPC completion time can increase by ~9.3x at 99^{th}.

the NSDI'21 website[2], a fairly light website composed of static pages. In Figures 3a and 3b, we show that performance variability in these applications is temporal. For instance, throughput varies about 45 Gbps in less than 100 μs.

Furthermore, in Figure 3c, we illustrate the CDF of throughput for both Memcached and Nginx. Again, we can see variability: although they can both reach 100 Gbps, but for 50% of the time their throughput stays under 80 Gbps and 60 Gbps for Memcached and Nginx, respectively.

Observation I: Slow receivers are pervasive and can manifest at short timescales.

There are many new congestion control algorithms. However, even new algorithms still suffer from slow receivers. To show this, we ran a number of tests using Homa, a state-of-the-art transport protocol for data center networks [72]. Precisely, we performed a few tests where a client requests Remote Procedure Calls (RPCs) on a server, a dominating pattern in production data centers [57, 86], using a workload similar to the one experienced by Memcached servers at Facebook [72].

In Figures 4a and Figure 4b, we show that when the endpoint cannot process incoming packets fast enough, the drop rate increases. In this experiment, all packet loss occurs at the end-host, and the core network is loss free. This is particularly problematic because packet reception is expensive [69] and CPU cycles spent to eventually drop a packet are wasted resources that can amplify the problem. For example, it has been demonstrated that an increasing loss rate can cause ad-

[2] https://www.usenix.org/conference/nsdi21

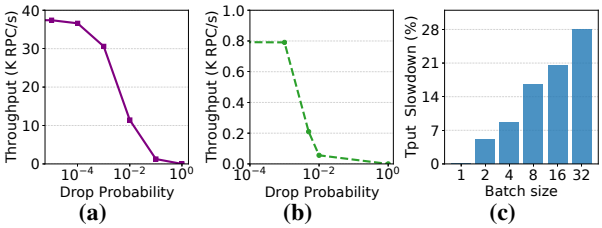


Figure 5: (a) The impact of packet loss on Homa's and, (b) TCP's throughput. Packet loss of 10^{-2} can halve the throughput. (c) The CPU cost of packet admission in PicNIC given different packet batch sizes. PicNIC's out-of-order packet completion queues incur high CPU utilization.

ditional CPU cycles spent in handling the transport protocol, leading to fewer available cycles for data processing [21].

Further, Figure 4c shows that slow receivers lead to an increase in RPC completion time. This is because vswitch queues are shared across flows/RPCs belonging to the same application. As a queue becomes full, flows not responsible for the congestion (victim flows) will experience high latency. Specifically, we can see that even a slight slow down of the receiver application can cause the 99.9^{th} percentile latency to hit values higher than 1ms. Those results show that even small amounts of packet loss can have a dramatic impact on the performance of receiver applications. We also performed a similar set of tests using DCTCP [12] to show DCTCP is also susceptible to high packet loss and report our results in Appendix (§ A.1.1).

To better understand the cost of packet loss, we performed an experiment where the vswitch is configured to drop packets according to a uniform probability distribution. We also used a standard TCP and Homa as transport protocols and measured the maximum sustainable throughput in terms of RPCs-per-second. In Figures 5a and 5b, we can see that even a *small percentages of packet loss* can dramatically impact performance. For example, the maximum sustainable RPCs-per-second can be halved with a packet loss probability of just 10^{-2} when using Homa. Also, the latency of Homa can reach milliseconds scale as packet loss exceeds 10^{-2}, as we show in Appendix (§A.1.2).

Observation II: Slow receivers cause sudden packet loss even in the presence of state-of-the-art congestion control mechanisms. Packet loss impacts network throughput and application service completion time.

2.2 Lossless Vswitching to the Rescue?

Packet loss at the vswitch is the source of many problems. However, there are already a variety of approaches that can be taken to avoid packet loss. These include reservations/rate-limiting, backpressure, credit-based flow control, or a combination thereof. Unfortunately, these have their own key limitations as discussed below and recap in Table 1.

Approach	Prevents packet loss	HOL blocking free	Avoids wasted CPU	Congestion spreading prevention
Rate-limiting [50]	✗	✗	✗	✗
Backpressure [31, 43]	✓	✗	✗	✗
Credit-based [62]	✓	✗	✓	✗
PicNIC [61]	✓	✗*	✗	✗
Backdraft	✓	✓	✓	✓

Table 1: A comparison of existing approaches to reducing packet loss. (*) PicNIC only prevents HOL blocking for flows coming from different VMs.

Reservation Schemes (Rate limiting). One option could be to rate-limit traffic according to bandwidth reservation schemes [13, 49, 50]. Although this is a good option for physical switches, it is not applicable in the virtual context. This is because such schemes assume that the line-rate processing is known in advance and deterministic. While this is the case for hardware switches, it is not for virtual ones.

Backpressure. Another option is to use a backpressure flow control scheme such as PFC [31] or BFC [43]. The main idea here is to send a pause message to the upstream switch before incurring a buffer overflow. Unfortunately, both PFC and BFC have key limitations that prevent them to be used as viable solution in a vswitch. The former might cause HOL blocking [29] and congestion spreading [44, 99] when the PAUSE frame from the vswitch reaches the upstream hardware switch. The second relies on the observation that most flows in a data center network are relatively short at today's 100 Gbps line-rates to avoid HOL blocking from priority hash collisions inside the network core. However, this assumption breaks if slow applications connected to a vswitch are allowed to generate PAUSE messages. In this case, slow receivers will cause congestion spreading, and hash collisions will result in reduced throughput of victim flows from line-rate (100 Gbps) to the rate of the slow receiver.

Credit-based Flow Control. Hop-by-hop credit-based flow control is another mechanism for ensuring zero packet drop [62]. Unfortunately, this technique requires an RTT to request credits and specific support from switches which makes it difficult to be deployed on production networks [24]. Similar to backpressure schemes, credit-based flow control requires packets to be buffered at switches when there are no credits available, leading to HOL blocking.

Observation III: Standard lossless techniques either cannot be used in a virtual context or cause severe HOL blocking and congestion spreading.

Other Approaches (PicNIC). PicNIC [61, 79] is a state-of-the-art solution to provide predictable performance in a multi-tenant data center where per-VM service level objectives (SLO) must be met. PicNIC takes an end-to-end approach to provide backpressure from receivers to senders and aims at preventing HOL blocking at the transmit-side by introducing a packet admission control system where descriptors may be completed out-of-order. This is implemented using a specific

Backdraft Component	Purpose	Expected result
Dynamic per-flow queuing	Avoids HOL blocking, On-demand memory usage	Mitigates tail latency, Improves throughput, Flexible packet scheduling, Prevents pause frame flood.
Doorbell queue	Avoids wasted CPU	Avoids extra pause frame generation, Saves network bandwidth, Alleviates the slow receiver problem.
Virtual switch backpressure overlay network	Avoids packet loss, Vswitch-level flow control, PFC/BFC compatibility	Avoids extra pause frame generation, Saves network bandwidth, Alleviates the slow receiver problem.

Table 2: Backdraft's components and their contributions

feature available in virtio interface [10, 78]. To understand its associated cost, we conducted an experiment where two end-points are connected to a vswitch on the same host. Each end-point is assigned a single core. Then we experimented with both *out-of-order* and *in-order* completion queues in the vswitch. Figure 5c, depicts that the out-of-order packet completion approach is slower than in-order by 20% and 28% when using a batch size of 16 [68] and 32 [3], respectively. Further, this is a baseline with only one core, and these overheads increase with a larger number of cores and queues. Thus, irrespective of application behavior, PicNIC imposes a high toll on performance. Furthermore, while PicNIC can successfully provide predictable performance for flows generated by different VMs, it does not have any mechanisms to ensure isolation between flows coming from the same VM as the out-of-order completion queues have a per-VM granularity, meaning that the slow-receiver problem can still happen and affect all the flows within the same VM.

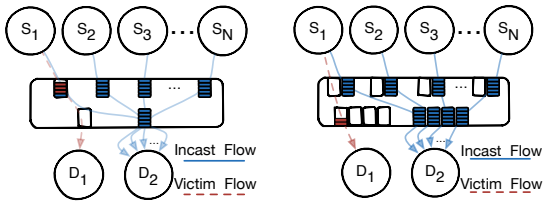
Observation IV: PicNIC can only isolate slow receivers at a per-VM granularity. It also imposes high CPU utilization and causes throughput degradation.

3 Backdraft Overview

Backdraft is a vswitch that provides lossless networking with higher throughput and lower CPU overheads than lossy switching, and Backdraft does not suffer from HOL blocking or congestion spreading. Backdraft achieves its goals by using three main components: (1) Dynamic Per-flow Queuing (DPFQ); (2) Separate queues for doorbells and data; and a (3) Virtual switch overlay network used for backpressure. Table 2 summarizes the purpose and effect of each component.

Dynamic Per-Flow Queuing (DPFQ): To avoid HOL blocking, Backdraft assigns a separate queue for every single flow in the vswitch, where a flow is an individual TCP connection. However, preallocating queues and memory for the worst case number of flows and burst sizes would be prohibitive. To ensure that per-flow queuing is not prohibitive in its memory overheads, we introduce a new approach that dynamically reclaims queues from idle flows and dynamically resizes queues to accommodate in-flight packets from bursty flows (DPFQ).

By enabling per-flow queueing, Backdraft fundamentally eliminates the HOL blocking caused by slow receivers and



(a) A traffic pattern where using backpressure suffers from HOL blocking.

(b) An illustration of why using separate queues for each virtual switch port avoids HOL blocking.

Figure 6: Queuing and HOL blocking with backpressure.

incasts. HOL blocking only occurs when flows share a queue, and every flow in DPFQ is served by its own queue (Figure 6b versus Figure 6a). DPFQ is possible because end-host memory is not as limited as in physical switches [43, 84]. However, the challenge is ensuring that DPFQ does not incur prohibitive memory overheads even though the number of active flows is potentially large [77]. Over-provisioning leads to memory pressure, while under-provisioning forces flows to fall back to sharing the same queue, potentially incurring HOL blocking.

To solve this issue, Backdraft introduces a new approach to efficiently resize queues on demand. Although all memory for queues and packet buffers is allocated when the process is created to avoid performance stalls, queues are dynamically allocated and reclaimed from flows as they start and stop, and queues are dynamically grown by combining queues as needed to accommodate bursts of packets. This dynamism allows for efficient per-flow queuing without increasing memory overheads. Our insight is that the total amount of congestion that can occur in a vswitch is limited by things like the line-rate of the NIC and not by the number of active flows. Given the same amount of memory, DPFQ enables the same congestion tolerance as a single queue.

DPFQ introduces a new interface to the vswitch. However, it is still possible to support DPFQ without modifying applications. For example, most TCP applications (*e.g.*, POSIX sockets applications) already perform per-flow operations. In this case, only the TCP stack needs to be modified to support DPFQ. Further, Backdraft supports legacy DPDK [47] and Netmap [76] applications that expect a shared queue interface with a vswitch by performing DPFQ inside the vswitch.

Doorbell Queues: The CPU overheads of a vswitch increase linearly with the number of queues that need to be polled [41], and data center workloads may have thousands of flows [18, 77]. Backdraft overcomes this limitation by using separate queues for data and doorbells. For each endpoint, there is a data queue for each flow and a doorbell queue for each core. To send data, an end-point first enqueues packets in data queues then sends a doorbell message to the doorbell queue. This allows the vswitch to poll only an application's doorbell queue to learn about new data.

Doorbell queues also provide a mechanism for applications to communicate scheduling information about the rel-

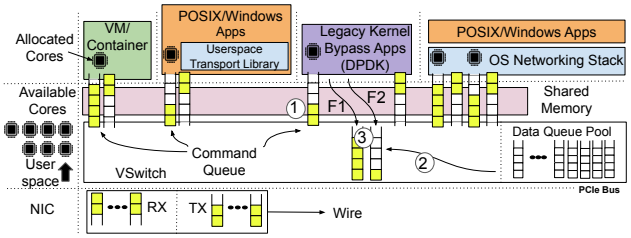


Figure 7: An overview of Backdraft's architecture.

ative priorities and weights of all active flows. Similar to prior work [79, 80, 87], this enables Backdraft to perform programmable scheduling and ensure that the appropriate queues are scheduled first to ensure low latency.

Virtual Switch Backpressure Overlay Network: When combined with backpressure, DPFQ can avoid both packet loss and HOL blocking for traffic local to the vswitch (server). However, if Backdraft runs out of buffer space and data is still incoming from a NIC, it must send a pause frame to the upstream TOR switch connected to the NIC to avoid packet loss when interfacing with a lossless network core, and it must drop packets when interfacing with a lossy network core. Unfortunately, generating pause frames can lead to congestion spreading, while dropping packets has a significant impact on network performance (Figure 5a and Figure 5b).

To avoid such problems, Backdraft builds an overlay network out of vswitches where Backdraft eagerly sends pause messages on the overlay network to the upstream vswitches that are causing congestion and either lazily sends pause messages to the upstream physical switch or lazily drops packets. This enables the local congested vswitch to continue buffering packets while waiting for the remote vswitch to react without causing congestion spreading. Additionally, DPFQ ensures that there is no congestion spreading inside the upstream vswitches because it is possible to pause only the flows responsible for the congestion.

With a lossless network core, the difference between the overlay pause threshold (Th_{over}) and the network pause threshold used for PFC or BFC (Th_{net}) determines the amount of data that can be buffered while waiting for the upstream vswitch to react. If the difference in bytes between these two thresholds is greater than the current network's bandwidth delay product (BDP), *i.e.*, the RTT times the network line rate ($Th_{over} - Th_{net} > RTT * BW$), then it is possible for the overlay network to react to a slow receiver without needing to send a network-level pause message. Because buffering 1ms of packets at 100 Gbps line-rate only requires 12.5 MB of buffering, it is easy to buffer multiple BDPs of packets in a vswitch with low overheads.

4 Design

Applications connect to Backdraft through queues implemented on top of shared memory, and both applications and

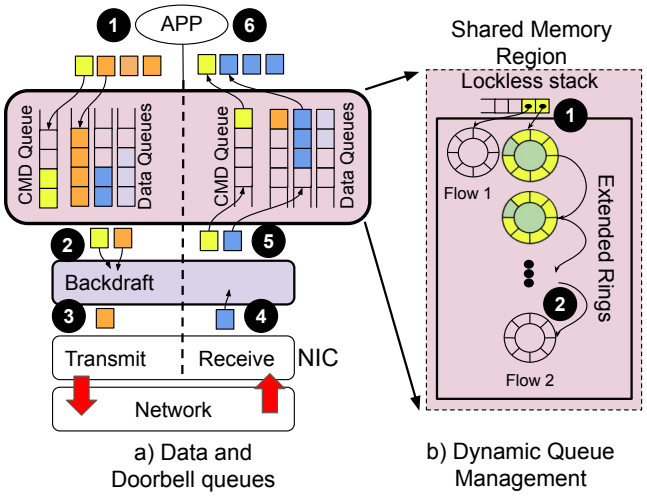


Figure 8: (a) Life cycle of control messages and data messages. (b) Data queue pool memory overview in DPFQ.

the vswitch detect packets by polling. Native Backdraft applications use doorbell queues and data queues in both RX and TX directions. However, Backdraft also supports legacy DPDK applications that only use data queues to send packets as well as standard applications using the kernel networking stack through a custom kernel driver. Currently, Backdraft is designed to be a userspace vswitch although its key ideas are also applicable to kernel-space switching.

Figure 7 provides an overview of Backdraft. First an application sends control messages to the vswitch (1). Upon the arrival of the doorbell message, Backdraft allocates the appropriate data queues (2). Finally, data packets exchange starts (3). Similarly, as new flows start and stop, an application can send more doorbell messages to allocate or release additional data queues. Backdraft supports dynamic queue allocation and resizing via a linked list structure to efficiently manage packet buffers. The rest of this section discusses the design of the Backdraft components in more detail.

4.1 Doorbell Queues

Backdraft uses doorbell queues to reduce the CPU overheads of DPFQ and achieve high throughput. DPFQ increases the number of available queues, and polling them all is inefficient. Checking for outstanding packets costs a memory access, which requires ~100 cycles per queue. There are two ways doorbell queues reduce polling overheads: First, only doorbell queues and not data queues need to be polled. Second, the total number of doorbell queues is kept small. To support parallelism, an application needs at most one hardware thread per doorbell queue.

Figure 8a illustrates the control flow between doorbell and data queues. (1) The application generates a doorbell message, notifying the vswitch. (2) The vswitch receives outstanding packets. If the destination is a remote server, (3) the vswitch sends the packets to the NIC, and then (4) the packets arrive

at the destination vswitch. Once the packet is at the receiving vswitch, ⑤ the vswitch places the received packet in the appropriate per-flow queue and then generates doorbell message for the application. Finally, ⑥ the application receives a doorbell message and then polls the data. Additionally, all of the command messages in a command queue are read at once in a batch to ensure there is no HOL blocking.

4.2 Dynamic Per-Flow Queuing (DPFQ)

It is important to ensure that DPFQ does not put pressure on memory; hence, DPFQ dynamically reclaims, reassigns, and resizes queues to reduce the memory pressure.

When an application initially connects to Backdraft, the data queue descriptors are negotiated between the vswitch and the application. As applications push packets to buffers, Backdraft allocates individual queues on demand (Figure 8b). To prevent applications' address spaces from being exposed to others, separate shared memory regions and pools of queues are used for each application. This separation of buffering across applications ensures isolation.

Backdraft dynamically resizes queues to absorb packet bursts while minimizing memory overheads. To this end, Backdraft allocates ring buffers of fixed size and then links them together to form and extend queues (Figure 8b-2). Before a ring buffer gets full, Backdraft extends the queue by placing a pointer to a new ring buffer instead of a packet buffer in the overloaded queue. This enables it to learn about an extended queue without any race conditions. Then, once a flow becomes idle, Backdraft reclaims the initial queue into a pool that it can allocate to other queues.

Backdraft pre-allocates all queues at boot time and pushes all the pointers to these queues in a lockless stack. The number of pre-allocated queues can be configured depending on the workload but we used 50 queues for the experiments of this paper. Backdraft benefits from the lockless stack in two ways: First, this structure improves cache efficiency as a pushed pointer can be used immediately from the top of the stack. Second, Backdraft is capable of supporting multiple threads accessing the data queue pool. When a new flow arrives at Backdraft, it borrows a pointer to a queue from the stack and enqueues packet pointers in the queue (Figure 8b-1). If this queue becomes fully occupied, Backdraft borrows another pointer and links it to the previous one as is depicted in Figure 8b-2.

Backdraft does not deallocate empty queues, nor does it leave empty queues allocated to idle flows. Instead, it reclaims empty queues and pushes them back to the lockless stack. This helps Backdraft to reuse reclaimed queues promptly without deallocating them. Backdraft is only responsible for queue assignment/reclamation leading to no race conditions. An entire queue can be reclaimed once there are no outstanding packets in the queue. Full reclamation only happens when a receiver application notifies Backdraft by means of a doorbell message about the emptiness of a data queue. Similarly, for new queues, applications must send a doorbell message to Backdraft requesting a queue corresponding to the new flow.

Both RX queues and TX queues can be extended. RX queues are frequently extended to tolerate bursts. In contrast, TX queues are only extended for flows with large BDPs, and there is no need to extend TX queues beyond a BDP in length. Instead of extending transmit queues beyond a BDP in length, an application can infer that a transmit queue being full is because of congestion or a slow receiver, and DPFQ enables applications to react to congestion. Many applications can simply keep packets buffered inside a TCP stack until the queue drains. However, it is also possible for some applications to mutate or even discard packets to reduce load.

Legacy Interfaces: Backdraft is backward compatible with both POSIX applications and DPDK applications. For the former, there are two ways to interface with Backdraft: (1) Backdraft uses a userspace TCP library that dynamically links to legacy socket applications (TAS [56]). (2) Packets can be received from the kernel through a custom networking driver. This is useful for applications that require features not yet supported by our library, *e.g.*, `PF_RING`.

4.3 VSwitch Backpressure Overlay Network

When there is congestion because of a slow receiver, Backdraft uses backpressure and sends pauses messages to the upstream sources of traffic to avoid packet losses. However, Backdraft is unique in that there are two different types of pause messages that it can generate: Overlay Pause Frames (OPFs) that are sent on a vswitch-to-vswitch overlay network and network-level pause frames that are sent hop-by-hop across the physical topology by a backpressure flow control scheme like PFC or BFC [43]. Backdraft implements PAUSEs internally by function calls instead of sending PAUSE frames throughout the pipeline because this reduces CPU overheads. PAUSE frames are only created if the PAUSE frame is destined for a remote end-point, which enables Backdraft to provide lossless forwarding across a cluster.

To avoid congestion spreading, Backdraft eagerly generates OPFs. When the occupancy of a receive queue crosses a configurable threshold (Th_{over}), Backdraft generates an OPF and sends it to the upstream Backdraft virtual switch that is causing congestion. Because there is only one flow per receive queue in Backdraft, only one message needs to be generated.

OPFs contain three pieces of information that are used by the upstream vswitch: 1) flow identifier, 2) pause time, and 3) new transmission rate. When an upstream vswitch receives an OPF, it pauses the input queue for the specified pause time, and then it applies a transmission rate-limit on the input queue.

Although prior backpressure schemes only send a pause time, sending a rate in an OPF is important to avoid persistent on/off congestion bursts from transmitters restarting after

being paused. To support this, Backdraft tracks an estimated receive rate (R_{recv}) using an exponential weighted moving average (EWMA) for each receive queue as it delivers packets, and it uses this rate when generating an OPF. The pause time is set as $(Th_{curr} - Th_{goal})/R_{recv}$ where Th_{curr} is the current length of the queue and Th_{goal} is the target queue length, which is equal to the batch size of packets read by the TCP stack by default to help ensure efficient CPU utilization.

The biggest concerns with respect to choosing values for Th_{goal} and Th_{over} are in avoiding starvation and reducing CPU overheads. Starvation is possible if the receiver vswitch either underestimates the end-point's rate or sets too large of a pause value. Th_{goal} provides headroom to avoid this, and if starvation is observed to be a problem with a running application, both the application and the vswitch can increase this value. In contrast, to avoid congestion spreading, it is desirable to set as large of a value for Th_{net} as possible because Backdraft generates PFC/BFC messages that will be processed by the upstream switch when this threshold is exceeded. This value can be as large as the maximum length of the queue minus the 1-hop bandwidth-delay product between the server and its TOR switch (1-hop RTT $\times$ line-rate).

On the whole, sending OPF messages significantly reduces CPU utilization by preventing packet drops. However, to reduce the CPU overheads of OPF messages, Th_{over} is set to be at least one batch size of packets larger than Th_{goal} to not interfere with batching. Further, to avoid excessive OPF generation, Backdraft generates a new OPF message only if the previous OPF message pause time has gone past. When the pause time passes, Backdraft checks the queue length to decide whether to generate another OPF message or not.

5 Implementation

Backdraft builds upon the BESS virtual switch [3] (commit 0145a1c). We have extended the TAS TCP stack [56] (commit a1c158f) to support TCP legacy applications. Further, we have implemented a Homa open-loop app based on the Homa DPDK library (commit 392b577) and altered the DPDK driver to interface with Backdraft. Our changes to BESS amount to about 3.5K LOC, and our changes to TAS and Homa required about 100 and 500 LOC, respectively. Apps running TAS and Homa both connect to BESS via a DPDK vHost user port.

6 Evaluation

In this section, we evaluate the performance of Backdraft and demonstrate that Backdraft is able to prevent packet loss while providing 100 Gbps switching capabilities and without incurring in HOL blocking.

Experimental cluster: We used two different types of clusters from CloudLab [75]. On the first, we were able to use PFC to perform experiments with a lossless fabric. This cluster has 6 servers, and each server has an Intel Xeon E5-2640 CPU running at 2.40 GHz with 64 GB of RAM and a 10G ConnectX4-L NIC. These servers are connected via a Mellanox SN240 10 Gbps TOR switch. We used a second cluster with 4 servers to perform experiments at 100 Gbps. Each server has an AMD EPYC 7452 64-Core CPU running at 2.30 GHz with 128 GB of RAM and a 100 Gbps ConnectX-5 NIC. These servers are connected via a Dell Z9264F-ON switch.

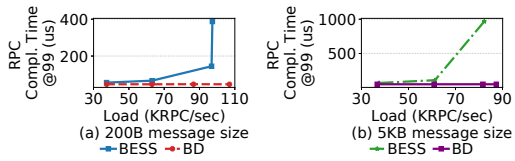


Figure 9: Performance of a victim RPC with Homa in the presence of an increasing load generated by a competing application. We used two different message sizes and either BESS or Backdraft as vswitch. The victim RPC is less impacted by the competing workload in the presence of Backdraft.

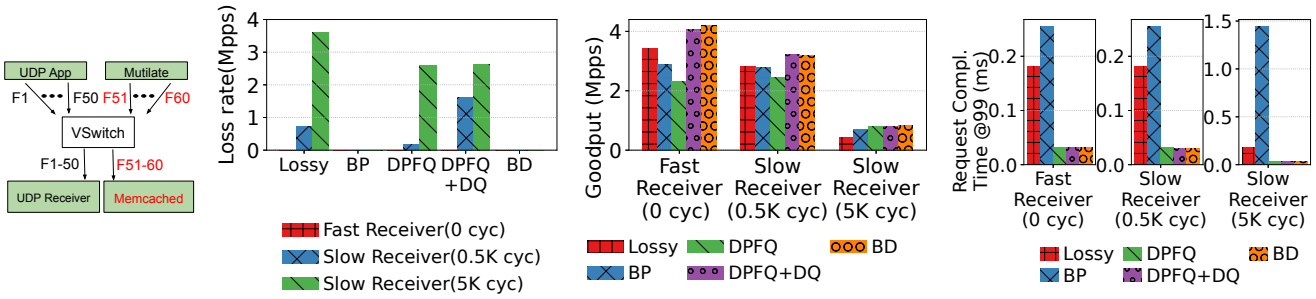
Applications: When experimenting with TCP, we leveraged the TAS TCP acceleration service to connect three unmodified POSIX applications to Backdraft: Memcached [7], Mutilate [64], and a custom distributed application that performs RPCs. To perform experiments with Homa, we utilized the Homa DPDK implementation [4], which unfortunately does not have any native support for applications. We overcome this problem by developing an open-loop RPC application on top of Homa. Because PicNIC [61] is proprietary software, a head-to-head comparison is not feasible.

Performance metrics and comparison points: Our experiments focus on four main metrics: packet drop rate, CPU utilization, throughput, and 99^{th} percentile request completion time latency. We also compared Backdraft against two variations of BESS virtual switch: lossy (default), and a lossless variation which generates PFC messages.

Key results: With Backdraft, the Homa-based RPC application achieves 20x lower tail latency at the 99^{th} percentile (§6.1). Further, Memcached achieves 1.9x higher goodput with Backdraft (§6.2). In a lossless multi-node scenario, Backdraft prevents congestion spreading in the network core (§6.3). In a 100 Gbps setup, Backdraft avoids HOL blocking and reaches 100 Gbps even in presence of slow receivers (§6.4). Finally, Backdraft supports 16 K queues without any throughput slow down (§6.5).

6.1 Backdraft Complements Homa

Our first experiment demonstrates that Backdraft complements Homa. In this experiment, we used two different machines in the 100 Gbps cluster: one of them hosting three client applications and the other two server applications. Each client/server application is assigned to a single CPU core. We used two clients to generate fixed-size RPC requests towards one server. The other client, instead, generates requests to-



(a) (b) (c) (d)

Figure 10: Performance of the individual components of Backdraft in presence of slow receivers when handling a Memcached TCP incast (10 flows) workload with a background UDP workload (50 flows). (a) Experimental setup (b) Aggregate drop rate, when the UDP server spends on average 0/0.5 K/5 K extra cycles on every delivered packet. Slower receivers have more detrimental impact on the performance. (c,d) detailed breakdown of goodput and latency impact of Backdraft. Backdraft improves tail latency up to 5.65x compared to BESS, and 45.2x compared to BESS augmented with PFC at the 99^{th} percentile while achieving 1.9x higher goodput.

wards the remaining server using the Facebook Memcached workload [15].

We compare the RPC performance of the client using the Memcached workload when using either BESS or Backdraft as vswitch. In Figure 9a and Figure 9b, we show the results when fixed-size RPCs are 200 B and 5 KB, respectively. When the RPC load increases, the completion time with Backdraft remains stable, while it inflates by over 20x with BESS. The poor results experienced with BESS are a consequence of its single queue design. In contrast, Backdraft keeps tail latency low because each `RPC_ID` occupies a single queue in the vswitch. This way, Backdraft removes HOL blocking of various RPCs with different service times.

Homa and Backdraft have strong synergy. Homa eagerly sends `RESEND` control messages to peers (`RESEND_INTERVAL = 2` μs [5]). This enables Homa to detect packet loss proactively, resulting in better tail latency. The CPU overhead of this task can be prohibitively high in presence of packet loss. For instance, without Backdraft, the CPU usage of Homa increases 8-10% because there are more outstanding messages to manage due to loss. Backdraft avoids wasting CPU cycles by preventing packet loss, enabling the transport protocol to provide better performance.

6.2 Per-Component Analysis

To provide a performance breakdown of the benefits of the different Backdraft'components, we created a scenario in a single host where background UDP packets destined to a slow receiver (50 flows) compete against a Memcached application with 10 active flows generated by Mutilate (Figure 10a). Here, we considered three cases: (1) the receiver spends 0 cycles processing the received packet; (2) the receiver spends 500 cycles; and (3) the receiver spends 5000 cycles. For context, Facebook's Katran load balancer spends 100 cycles per packet [20], and complex functions like range queries in key-value stores can easily take more than 1 K cycles.

Figure 10 shows the results of this experiment. With *Lossy*, we consider the default behavior of BESS, while *BP* is BESS with PFC enabled. DPFQ, DPFQ+DQ, and DPFQ+DQ+ON (BD) show the incremental benefits of different Backdraft components: dynamic per-flow input queueing (DPFQ), doorbell queues (DQ), and the overlay network (ON). BD indicates our final system with all components.

Figure 10b shows packet loss rates given the slow receiver application (UDP receiver) in Figure 10a. BESS with PFC and Backdraft both report zero packet loss. Without PFC for BESS and without the overlay network for Backdraft, packets may be dropped. Packet loss occurs in both the slow and fast flows, and it is more problematic in the presence of a slow receiver. DPFQ+DQ reduces CPU overheads and can forward at higher throughputs than just DPFQ. This results in even more packet loss at the receiver. This packet loss, however, is avoided by introducing the overlay network (ON). Backdraft prevents packet loss and achieves higher throughput and lower tail latency.

Next, Figure 10c shows the aggregate goodput achieved by the applications (UDP and Memcached). Backdraft always outperforms BESS, even when the latter is augmented with lossless capabilities using PFC. In this experiment, for the 0, 500, and 5 K cycle receiver, Backdraft achieves 22%, 10%, and 200% and higher goodput than the lossy counterpart, respectively. Backdraft also mitigates tail latency at the 99^{th} percentile by up to 5.65x.

Looking at the individual components, we find that DPFQ has a negative impact on performance because polling more queues consumes more CPU cycles. However, combining DPFQ and doorbell queues (DPFQ+DQ) improves goodput by reducing cycles spent polling. This effect is more visible in the presence of a fast receiver, as the faster the receiver the more packets need to be process by the vswitch. The last component (ON) enables Backdraft to prevent packet loss. This is illustrated in Figure 10b.

Figure 10d shows the latency experienced by Memcached. In this figure, Backdraft similarly outperforms both BESS configurations. The BP bar shows that naively applying a backpressure mechanism dramatically increases network latency, and this is mainly an effect of HOL blocking. DPFQ, in

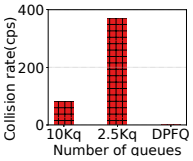


Figure 11: Compares collision rate of a skewed workload when varying the number of queues.

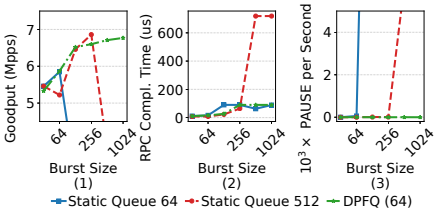


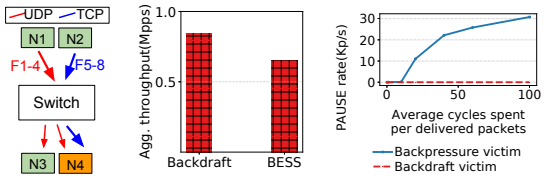
Figure 12: Shows goodput, latency and PAUSE rate of short queues, long queues verses DPFQ. Backdraft can absorb different burst sizes compared to static queue allocation with PFC.

contrast, keeps the overall latency low, even in the presence of a slow receiver. This is because providing per-flow queues prevents HOL blocking.

To better understand Dynamic Queue Allocation (DQA), we used a sample client application generating approximately 100 K flows to a server sink application on the same machine where only 1 K flows are active at any point in time. We compared two different policies for queue allocation: a static number defined at configuration time and a dynamic. The former assigns flows to queues using an RSS (Receive Side Scaling) hash function, the latter creates a new queue anytime a new flow shows up. Figure 11 shows that when using only 2.5 K queues, the collision rate is high, even if only 1 K flows are active. Having 10x more static queues than active flows helps, but still collisions occur. In contrast, DPFQ avoids wasted memory and achieves a zero-collision rate thanks to its per-flow queueing mechanism. Each ring buffer consumes about 20 B. 10 K queues will consume 200 KB, where DPFQ allocates only 1 K queues since we have 1 K active flows, requiring only 20 KB. This is a 10x reduction in memory utilization in addition to the reduction in collisions.

Next, we evaluated Backdraft's ability to absorb packet bursts by extending queues by performing an experiment where a sender pushes different batch sizes (64 to 1024) to a receiver. The receiver is attached to a vswitch on the same server and pulls packets in large batches of 1024. This experiment compares Backdraft against two different configurations of BESS augmented with PFC: one with short queues, the other with long. Short queues are more likely to generate PAUSE frames at a higher rate, whereas longer queues are less likely.

Figure 12 shows that, when increasing the burst size, lossless BESS with short queues causes a high PFC PAUSE frame generation rate that would hurt application performance in terms of goodput and tail latency. Although long queues reduce the PAUSE frame generation problem, this is at the cost of increased latency. In contrast, this experiment shows that Backdraft is able to absorb variations, particularly in a bursty workload with its dynamic queue extensions. It is the only configuration that does not generate PAUSE frames. Moreover, Backdraft maintains high goodput with DPFQ because the cost of queue extension is relatively low.



(a) (b) (c)

Figure 13: Performance of Backdraft overlay network in a cluster-wide experiment. Backdraft achieves higher throughput and avoids extra PAUSE frame generation in presence of a slow receiver. (a) Experiment setup. (b) Overall throughput. (c) Pause frame generation rate due to a slow receiver.

Config	Tput (Gbps)	Pause (Kfps)	Drop (Mpps)
BESS Lossy + Lossy Network	(2.36,21.85)	N/A	(1.6,0)
BESS Lossless + Lossy Network	(2.66,19.29)	(2.8,0)	(1.3,0)
ON + Lossy Network	(2.3,21.98)	(0,0)	(0,0)

Table 3: Virtual overlay network performance results (`Victim,Non-victim flow`)

Finally, we also measured the overheads of extending queues in DPFQ and found that it is small. The number of cycles required to extend queues fluctuates between 24 and 350 cycles, and this value is dependent on caching. This shows that the overheads of DPFQ are low, especially when amortized over all of the packets in the added queue, which has a default size of 64 packets. Further, if desired, Backdraft's queue size can be configured as a parameter based on the measured overhead according to the user's preference for performance versus memory efficiency.

6.3 Multi-node Performance

This section studies the behavior of the overlay network between vswitches used in Backdraft. To do this, we used a cluster of four different servers. Each server is running its own vswitch, and they are connected through a physical switch with PFC enabled. We generated background UDP flows competing with TCP victim flows (Figure 13a) and compared the results when using either BESS or Backdraft as a vswitch. Backdraft achieves higher aggregate throughput than BESS (Figure 13b). This is because Backdraft sends PAUSE frames through the overlay networks as soon as it notices queue buildup. This is not done by BESS, which in turn induces the physical switch to send PFC PAUSE frames and trigger congestion spreading inside the network.

Figure 13c shows the number of PFC PAUSE frames sent by the receiving server to the upstream PFC enabled switch as a receiver gets slower. In this scenario, BESS causes the physical switch to also generate PAUSE frames. However, this does not happen with Backdraft because the overlay network pauses the flow for the slow receiver before queues fill up.

We also compared the performance of BESS and Backdraft using two nodes in the 100 Gbps CloudLab testbed connected

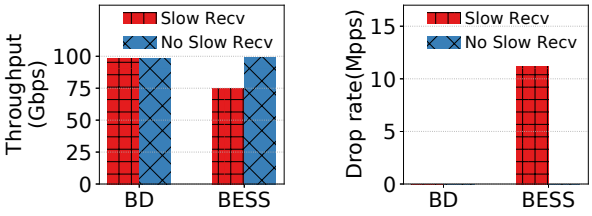


Figure 14: Multithreading in Backdraft, (a) Aggregate throughput, (b) Drop rate of victim flow. Backdraft achieves 100 Gbps using multiple cores while ensuring zero drop at the vswitch. In this experiment, applications are allocated enough number of cores to drive 100 Gbps.

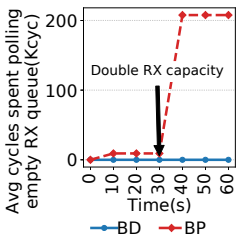


Figure 15: Backdraft has no starvation. Backdraft adjusts the sender rate using overlay messages to ensure enough buffering at the receiver's queues.

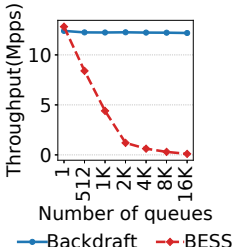


Figure 16: Backdraft, compared to BESS, sustains high throughput with a single core while managing large number of queues as it mitigates the polling overhead.

by a lossy switch. Here, we used one server to send two UDP flows towards another machine where one receiver is slow and the other instead is a victim. Table 3 reports our results. When using standard BESS (lossy) with a lossless network core (first row), the overall throughput is high. However, it also suffers from a high degree of packet loss. When, instead, using BESS generating PFC frames (lossless), the throughput is reduced and a considerable amount of packet loss still appear, as the network core is lossy. Finally, due to overlay messages, Backdraft it is able to avoid packet losses, while keeping network throughput high.

Finally, we performed an experiment to demonstrate that the overlay network in Backdraft does not suffer from starvation, even when the rate of the slow receiver is variable over the time. In this experiment, one machine is sending packets towards a slow receiver. Initially, the destination polls packets at rate 3 Mpps, then it doubles its rate at time T_{30}. In Figure 15, BP (BESS with PFC) suffers from starvation and the receiver spends its extra cycles polling instead of processing packets. In contrast, at $T = T_{30}$, Backdraft detects a change in the receivers rate and increases Th_{goal} to avoid starvation for the rest of the application's life.

6.4 100 Gbps Forwarding Performance

To show that Backdraft can achieve 100 Gbps throughput regardless of the presence of slow receivers, we performed an experiment where an 8-core sender is generating a heavily skewed workload consisting of 12 flows (11 fast flows and 1 slow flow) towards an 8-core receiver. To cause a slow flow, one of 8 cores of the receiver application is slowed down in this experiment. When using BESS, the slow flow will eventually block the others, forcing the vswitch to drop packets due to a lack of queue descriptors at the receiver's RX queues. In contrast, Backdraft does not suffer from this problem because of its ability to dynamically resize queues and send overlay PAUSE frames.

Figure 14 shows the aggregate throughput for all flows and the drop rate for the victim flow in presence of slow receivers in this experiment. With BESS, this experiment results in high packet loss and a decreased throughput of ~75 Gbps even though there is only one core receiving slower than the expected pace. In contrast, Backdraft achieves full line-rate without any packet drops. Backdraft sends overlay messages on a per-queue basis to notify the upstream sender to reduce its rates. This allows Backdraft to utilize the extra bandwidth for the other flows in order to drive the 100 Gbps line-rate.

6.5 Backdraft Scalability

Finally, we assessed the scalability of Backdraft in terms of its throughput and memory requirements.

The impact of number of queues on performance. To demonstrate the benefits of doorbell queues, we performed an experiment where an application sends packets from UDP flows in a skewed pattern based on the Zipfian distribution, and we compared the throughput achieved between doorbell queues (Backdraft) and polling every queue (BESS).

Figure 16 shows the aggregate vswitch throughput when a single core is allocated to the switch as we vary the number queues. With a small number of queues, both Backdraft and BESS perform similarly, which shows that the overheads of doorbell queues are quite low. However, when the number of queues increase, only Backdraft maintains its throughput.

The amount of memory needed varying network RTT. Finally, we performed an analysis of the memory overheads of Backdraft to demonstrate that this is not prohibitive. In order to avoid congestion spreading, Th_{net} must be sufficiently larger when compared to Th_{over} so that packets can be buffered during the time it takes for the source of the congestion to pause and adjust its rate. The increased memory overheads of Backdraft are small and can be estimated by bandwidth-delay product for different network line-rates. For example, a 100 Gbps network with a 1ms RTT only requires 12.5 MB of buffering to avoid congestion spreading. Further, it is important to note that DPFQ ensures that this buffering requirement is for the entire switch and not per-flow.

7 Discussion

Slow NICs. NICs may be slow and unable to achieve line-rate, and this can cause packet loss [44, 83]. If a slow NIC

participates in the overlay network by generating OPFs, it can avoid both packet loss and congestion spreading.

Slow virtual switches. There are many reasons a vswitch may be slow, including CPU limitations, memory bandwidth limitations, and insufficient LLC cache [34, 36, 89, 95], and packets can be dropped at the NIC when a vswitch cannot keep up with the ingress rate, resulting in a slow vswitch problem. This can be solved by offloading part of Backdraft's processing onto a programmable NIC [2, 9, 39, 46, 87]. This should be feasible because recent developments in NIC designs have brought models that provide a large amount of on-NIC memory that can be used for Backdraft. For example, Xilinx Alveo NICs support High Bandwidth Memory (HBM), fast memory that is directly embedded on-chip in an FPGA [11, 53].

RDMA support. Backdraft can support 2-sided RDMA verbs by monitoring the length of receive queues in the application or a library and generating OPF messages to transmitters as appropriate. Further, if offloaded onto a NIC, Backdraft can mitigate the effect of the slow NIC problem for 1-sided verbs and complement the sender-based approaches that can be used for congestion control [55, 68, 98].

Programmable packet scheduling. If Backdraft is deployed without enough memory, multiple flows have to share the same queue. Although this can cause HOL blocking, this can be mitigated with opportunistic packet scheduling. For example, Backdraft could employ software solutions like Eiffel [80] or hardware ones like AIFO [94] and PIFO [85] if Backdraft is offloaded to a programmable NIC.

Linux kernel compatibility. Backdraft is implemented using DPDK. Thus, all of the traffic coming from the NIC bypasses the Linux kernel. However, we believe that the same design principles are applicable to the Linux kernel. Further, being implemented in userspace does not even preclude Backdraft from interfacing with the Linux kernel networking stack. For example, Backdraft can use a custom kernel driver to interface with traditional applications, and the recently proposed `AF_XDP Poll Mode` driver [1] enables DPDK applications to natively support the AF_XDP socket and retain compatibility with the Linux tools that operators expect [90].

8 Related Work

Slow receiver problem. Past research has acknowledged the slow receiver problem in the context of the overheads of the Linux networking stack [21], Linux-based transport protocol implementations [73], and production networks from Microsoft [44], and Google Swift [60].

Virtual switching. Snap, Andromeda, and PicNIC all perform lossy vswitching [30, 61, 68], which drops packets. On the other hand, Zfabric, NFVNice, and zOVN are lossless vswitches. These, however, suffer from HOL blocking as they share queues among active flows in the vswitch [26, 27, 59]. Moreover, unlike Backdraft, none of these approaches address the slow receiver problem. Similarly, FreeFlow ensures high performance by using shared memory, but it does not consider packet loss problem at the end-hosts [93].

Packet scheduling and rate limiting. Backdraft is compatible with Eiffel and Carousel and can mitigate their CPU utilization overheads with its command queue [79, 80]. Similarly, hyperplane can be used to reduce the CPU polling overheads of Backdraft [70]. EyeQ is a related system that builds an overlay network that performs rate-limiting [50]. However, EyeQ pays high CPU utilization overhead when rate limiting, and EyeQ works at millisecond-scale, which is not fast enough to address the slow receiver problem.

Congestion control. In addition to Homa, there are other important new congestion control algorithms like Google's Swift, which performs fine grain time stamping to identify the congestion source (end-host, network) [60]. Similar to how we have found that Backdraft is complementary to Homa, we expect that Backdraft is complementary to Swift as well.

Flow control. Backdraft is complementary to flow control protocols designed to provide a lossless core network. For example, Backdraft is complementary to PFC because it strives to minimize the PAUSE frames sent across the network. PCN ensures high throughput for victim flows if congestion spreading occurs and is also complementary to Backdraft [23]. BFC is a new backpressure flow control protocol intended to replace PFC [43]. Backdraft solves a key problem that arises with deploying BFC in practice. This is because BFC assumes that flows can be received at 100 Gbps line-rates, and this assumption can be violated by slow receivers. Backdraft addresses this problem and prevents congestion spreading from slow receivers.

9 Conclusions

In this paper, we present the design and implementation of Backdraft, a new lossless virtual switch. We make a case for providing lossless networking at the vswitch level by showing the impact of packet loss caused by slow receivers on network performance using existing congestion control algorithms.

We implemented Backdraft on top of the BESS virtual switch and performed experiments with two different clusters of servers on CloudLab (10 Gbps and 100 Gbps). We used unmodified POSIX applications with TAS TCP and a custom distributed application that performs RPCs with Homa, a state-of-the-art datacenter transport protocol. We demonstrate that Backdraft is effective in preventing packet loss and reduces tail latency by up to 20x compared to BESS.

Acknowledgements: We thank our shepherd, Anurag Khandelwal, the anonymous NSDI reviewers, Praveen Kumar, and Djordje Jevdjic for their feedback. This work was funded by NSF Awards CNS-2200783 and CNS-2008273, the UK EPSRC project EP/T007206/1, and by gifts from Google and VMware.

References

[1] Af-xdp poll mode driver. https://doc.dpdk.org/guides/nics/af_xdp.html.

[2] Alveo SN1000 SmartNIC Accelerator Card. https://www.xilinx.com/products/boards-and-kits/alveo/sn1000.html.

[3] BESS: Berkeley Extensible Software Switch. https://github.com/NetSys/bess.

[4] Homa. https://github.com/PlatformLab/Homa.

[5] Homa commit 47265bf. https://github.com/PlatformLab/Homa/blob/392b577bbdad2f5aa42faefc88614992b5e505d2/src/TransportImpl.cc#L36.

[6] iperf3: Documentation. http://software.es.net/iperf/.

[7] Memcached. https://memcached.org/.

[8] Nginx. https://www.nginx.com/.

[9] Virtual Switch on BlueField SmartNIC. https://docs.mellanox.com/display/BlueFieldSWv20110841/Virtual+Switch+on+BlueField+SmartNIC.

[10] What's New in Virtio 1.1. https://www.dpdk.org/wp-content/uploads/sites/35/2018/09/virtio-1.1_v4.pdf.

[11] Xilinx alveo u280. https://www.xilinx.com/products/boards-and-kits/alveo/u280.html.

[12] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan. Data center tcp (dctcp). In *Special Interest Group on Data Communication (SIGCOMM)*. ACM, 2010.

[13] M. Alizadeh, A. Kabbani, T. Edsall, B. Prabhakar, A. Vahdat, and M. Yasuda. Less is more: Trading a little bandwidth for ultra-low latency in the data center. In *Networked Systems Design and Implementation (NSDI)*. USENIX, 2012.

[14] N. Amit, A. Tai, and M. Wei. Don't shoot down tlb shootdowns! In *European Conference on Computer Systems (EuroSys)*. ACM, 2020.

[15] B. Atikoglu, Y. Xu, E. Frachtenberg, S. Jiang, and M. Paleczny. Workload analysis of a large-scale key-value store. In *SIGMETRICS/PERFORMANCE Joint International Conference on Measurement and Modeling of Computer Systems*. ACM, 2012.

[16] L. Barroso, M. Marty, D. Patterson, and P. Ranganathan. Attack of the killer microseconds. In *Communications of the ACM*, volume 60, pages 48–54. ACM, 2017.

[17] L. A. Barroso, J. Clidaras, and U. Hölzle. *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines, Second Edition*. 2013.

[18] T. Benson, A. Akella, and D. A. Maltz. Network Traffic Characteristics of Data Centers in the Wild. In *Special Interest Group on Data Communication (SIGCOMM)*. ACM, 2010.

[19] J. Bouron, S. Chevalley, B. Lepers, W. Zwaenepoel, R. Gouicem, J. Lawall, G. Muller, and J. Sopena. The battle of the schedulers: Freebsd ULE vs. linux CFS. In *Annual Technical Conference (ATC)*. USENIX, 2018.

[20] M. S. Brunella, G. Belocchi, M. Bonola, S. Pontarelli, G. Siracusano, G. Bianchi, A. Cammarano, A. Palumbo, L. Petrucci, and R. Bifulco. hXDP: Efficient software packet processing on FPGA nics. In *Operating Systems Design and Implementation (OSDI)*. USENIX, 2020.

[21] Q. Cai, S. Chaudhary, M. Midhul, Vuppalapati, J. Hwang, and R. Agarwal. Understanding Host Network Stack Overheads. In *Special Interest Group on Data Communication (SIGCOMM)*. ACM, 2021.

[22] Y. Chen, R. Griffith, J. Liu, R. H. Katz, and A. D. Joseph. Understanding TCP incast throughput collapse in datacenter networks. In *Workshop on Research on Enterprise Networking (WREN)*. ACM, 2009.

[23] W. Cheng, K. Qian, W. Jiang, T. Zhang, and F. Ren. Re-architecting congestion management in lossless Ethernet. In *Networked Systems Design and Implementation (NSDI)*. USENIX, 2020.

[24] I. Cho, K. Jang, and D. Han. Credit-scheduled delay-bounded congestion control for datacenters. In *Special Interest Group on Data Communication (SIGCOMM)*. ACM, 2017.

[25] C. Chou, L. N. Bhuyan, and D. Wong. μdpm: Dynamic power management for the microsecond era. In *International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2019.

[26] D. Crisan, R. Birke, N. Chrysos, C. Minkenberg, and M. Gusat. zFabric: How to virtualize lossless Ethernet? In *International Conference On Cluster Computing (CLUSTER)*. IEEE, 2014.

[27] D. Crisan, R. Birke, G. Cressier, C. Minkenberg, and M. Gusat. Got loss? get zOVN! In *Special Interest Group on Data Communication (SIGCOMM)*. ACM, 2013.

[28] A. Daglis, M. Sutherland, and B. Falsafi. RPCValet: Ni-driven tail-aware balancing of μs-scale RPCs. In *Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. ACM, 2019.

[29] W. J. Dally and C. L. Seitz. Deadlock-free message routing in multiprocessor interconnection networks. In *Transactions on Computers*, volume 36, pages 547–553. IEEE, 1987.

[30] M. Dalton, D. Schultz, J. Adriaens, A. Arefin, A. Gupta, B. Fahs, D. Rubinstein, E. C. Zermeno, E. Rubow, J. A. Docauer, et al. Andromeda: Performance, isolation, and velocity at scale in cloud network virtualization. In *Networked Systems Design and Implementation (NSDI)*. USENIX, 2018.

[31] C. DeSanti. IEEE 802.1: 802.1Qbb - Priority-based Flow Control. http://www.ieee802.org/1/pages/802.1bb.html, 2009.

[32] D. Didona and W. Zwaenepoel. Size-aware sharding for improving tail latencies in in-memory key-value stores. In *Networked Systems Design and Implementation (NSDI)*. USENIX, 2019.

[33] C. Fang, H. Liu, M. Miao, J. Ye, L. Wang, W. Zhang, D. Kang, B. Lyv, P. Cheng, and J. Chen. Vtrace: Automatic diagnostic system for persistent packet loss in cloud-scale overlay network. In *Special Interest Group on Data Communication (SIGCOMM)*. ACM, 2020.

[34] A. Farshin, T. Barbette, A. Roozbeh, G. Q. Maguire Jr., and D. Kostić. PacketMill: Toward per-core 100-gbps networking. In *Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. ACM, 2021.

[35] A. Farshin, A. Roozbeh, G. Q. M. Jr., and D. Kostić. Reexamining direct cache access to optimize i/o intensive applications for multi-hundred-gigabit networks. In *Annual Technical Conference (ATC)*. USENIX, 2020.

[36] A. Farshin, A. Roozbeh, G. Q. Maguire Jr, and D. Kostić. Reexamining direct cache access to optimize I/O intensive applications for multi-hundred-gigabit networks. In *Annual Technical Conference (ATC)*. USENIX, 2020.

[37] M. Ferdman, A. Adileh, O. Kocberber, S. Volos, M. Alisafaee, D. Jevdjic, C. Kaynak, A. D. Popescu, A. Ailamaki, and B. Falsafi. Clearing the clouds: A study of emerging scale-out workloads on modern hardware. In *Special Interest Group on Programming Languages (SIGPLAN)*. ACM, 2012.

[38] D. Firestone. VFP: A virtual switch platform for host SDN in the public cloud. In *Networked Systems Design and Implementation (NSDI)*. USENIX, 2017.

[39] D. Firestone, A. Putnam, S. Mundkur, D. Chiou, A. Dabagh, M. Andrewartha, H. Angepat, V. Bhanu, A. Caulfield, E. Chung, H. K. Chandrappa, S. Chaturmohta, M. Humphrey, J. Lavier, N. Lam, F. Liu, K. Ovtcharov, J. Padhye, G. Popuri, S. Raindel, T. Sapre, M. Shaw, G. Silva, M. Sivakumar, N. Srivastava, A. Verma, Q. Zuhair, D. Bansal, D. Burger, K. Vaid, D. A. Maltz, and A. Greenberg. Azure accelerated networking: SmartNICs in the public cloud. In *Networked Systems Design and Implementation (NSDI)*. USENIX, 2018.

[40] J. Fried, Z. Ruan, A. Ousterhout, and A. Belay. Caladan: Mitigating interference at microsecond timescales. In *Symposium on Operating Systems Design and Implementation (OSDI)*. USENIX, 2020.

[41] H. Golestani, A. Mirhosseini, and T. F. Wenisch. Software data planes: You can't always spin to win. In *Symposium on Cloud Computing (SoCC)*. ACM, 2019.

[42] R. Gouicem, D. Carver, J.-P. Lozi, J. Sopena, B. Lepers, W. Zwaenepoel, N. Palix, J. Lawall, and G. Muller. Fewer cores, more hertz: Leveraging high-frequency cores in the OS scheduler for improved application performance. In *Annual Technical Conference (ATC)*. USENIX, 2020.

[43] P. Goyal, P. Shah, K. Zhao, G. Nikolaidis, M. Alizadeh, and T. E. Anderson. Backpressure flow control. In *Networked Systems Design and Implementation (NSDI)*. USENIX, 2022.

[44] C. Guo, H. Wu, Z. Deng, G. Soni, J. Ye, J. Padhye, and M. Lipshteyn. RDMA over Commodity Ethernet at Scale. In *Special Interest Group on Data Communication (SIGCOMM)*. ACM, 2016.

[45] S. Han, K. Jang, A. Panda, S. Palkar, D. Han, and S. Ratnasamy. SoftNIC: A software nic to augment hardware. *EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2015-155*, 2015.

[46] J. T. Humphries, K. Kaffes, D. Mazières, and C. Kozyrakis. Mind the gap: A case for informed request scheduling at the nic. In *Workshop on Hot Topics in Networks (HotNets)*. ACM, 2019.

[47] D. Intel. Data plane development kit, 2014.

[48] C. Iorgulescu, R. Azimi, Y. Kwon, S. Elnikety, M. Syamala, V. Narasayya, H. Herodotou, P. Tomita, A. Chen, J. Zhang, and J. Wang. Perfiso: Performance isolation for commercial latency-sensitive services. In *Annual Technical Conference (ATC)*. USENIX, 2018.

[49] K. Jang, J. Sherry, H. Ballani, and T. Moncaster. Silo: Predictable Message Latency in the Cloud. In *Special Interest Group on Data Communication (SIGCOMM)*. ACM, 2015.

[50] V. Jeyakumar, M. Alizadeh, D. Mazières, B. Prabhakar, C. Kim, and A. Greenberg. EyeQ: Practical Network Performance Isolation at the Edge. In *Networked Systems Design and Implementation (NSDI)*. USENIX, 2013.

[51] K. Kaffes, T. Chong, J. T. Humphries, A. Belay, D. Mazières, and C. Kozyrakis. Shinjuku: Preemptive scheduling for μsecond-scale tail latency. In *Networked Systems Design and Implementation (NSDI)*. USENIX, 2019.

[52] K. Kaffes, J. T. Humphries, D. Mazières, and C. Kozyrakis. Syrup: User-defined scheduling across the stack. In *Symposium on Operating Systems Principles (SOSP)*. ACM, 2021.

[53] K. Kara, C. Hagleitner, D. Diamantopoulos, D. Syrivelis, and G. Alonso. High bandwidth memory on FPGAs: A data analytics perspective. In *International Conference on Field-Programmable Logic and Applications (FPL)*. IEEE, 2020.

[54] H. Kasture, D. B. Bartolini, N. Beckmann, and D. Sanchez. Rubik: Fast analytical power management for latency-critical systems. In *International Symposium on Microarchitecture (MICRO)*. IEEE/ACM, 2015.

[55] G. P. Katsikas, T. Barbette, M. Chiesa, D. Kostić, and G. Q. Maguire. What you need to know about (smart) network interface cards. In *International Conference on Passive and Active Network Measurement*. Springer, 2021.

[56] A. Kaufmann, T. Stamler, S. Peter, N. K. Sharma, A. Krishnamurthy, and T. Anderson. TAS: TCP acceleration as an OS service. In *European Conference on Computer Systems (EuroSys)*. ACM, 2019.

[57] M. Kogias, G. Prekas, A. Ghosn, J. Fietz, and E. Bugnion. R2P2: Making RPCs first-class datacenter citizens. In *Annual Technical Conference (ATC)*. USENIX, 2019.

[58] A. Kopytov. Sysbench: a system performance benchmark. *http://sysbench. sourceforge. net/*, 2004.

[59] S. G. Kulkarni, W. Zhang, J. Hwang, S. Rajagopalan, K. K. Ramakrishnan, T. Wood, M. Arumaithurai, and X. Fu. NFVnice: Dynamic Backpressure and Scheduling for NFV Service Chains. In *Special Interest Group on Data Communication (SIGCOMM)*. ACM, 2017.

[60] G. Kumar, N. Dukkipati, K. Jang, H. M. G. Wassel, X. Wu, B. Montazeri, Y. Wang, K. Springborn, C. Alfeld, M. Ryan, D. Wetherall, and A. Vahdat. Swift: Delay is simple and effective for congestion control in the datacenter. In *Special Interest Group on Data Communication (SIGCOMM)*. ACM, 2020.

[61] P. Kumar, N. Dukkipati, N. Lewis, Y. Cui, Y. Wang, C. Li, V. Valancius, J. Adriaens, S. Gribble, N. Foster, and A. Vahdat. Picnic: Predictable virtualized nic. In *Special Interest Group on Data Communication (SIGCOMM)*. ACM, 2019.

[62] H. T. Kung, T. Blackwell, and A. Chapman. Credit-based flow control for ATM networks: Credit update protocol, adaptive credit allocation and statistical multiplexing. In *Special Interest Group on Data Communication (SIGCOMM)*. ACM, 1994.

[63] B. Lepers, R. Gouicem, D. Carver, J.-P. Lozi, N. Palix, M.-V. Aponte, W. Zwaenepoel, J. Sopena, J. Lawall, and G. Muller. Provable multicore schedulers with ipanema: Application to work conservation. In *European Conference on Computer Systems (EuroSys)*. ACM, 2020.

[64] J. Leverich and C. Kozyrakis. Reconciling high server utilization and sub-millisecond quality-of-service. In *European Conference on Computer Systems (EuroSys)*. ACM, 2014.

[65] J. Li, N. K. Sharma, D. R. K. Ports, and S. D. Gribble. Tales of the tail: Hardware, os, and application-level sources of tail latency. In *Symposium on Cloud Computing (SoCC)*. ACM, 2014.

[66] D. Lo, L. Cheng, R. Govindaraju, P. Ranganathan, and C. Kozyrakis. Heracles: Improving resource efficiency at scale. In *International Symposium on Computer Architecture (ISCA)*. ACM, 2015.

[67] A. Manousis, R. A. Sharma, V. Sekar, and J. Sherry. Contention-aware performance prediction for virtualized network functions. In *Special Interest Group on Data Communication (SIGCOMM)*. ACM, 2020.

[68] M. Marty, M. de Kruijf, J. Adriaens, C. Alfeld, S. Bauer, C. Contavalli, M. Dalton, N. Dukkipati, W. C. Evans, S. Gribble, et al. Snap: a microkernel approach to host networking. In *Symposium on Operating Systems Principles (SOSP)*. ACM, 2019.

[69] A. Menon and W. Zwaenepoel. Optimizing TCP receive performance. In *Annual Technical Conference (ATC)*. USENIX, 2008.

[70] A. Mirhosseini, H. Golestani, and T. F. Wenisch. Hyperplane: A scalable low-latency notification accelerator for software data planes. In *International Symposium on Microarchitecture (MICRO)*. IEEE/ACM, 2020.

[71] A. Mirhosseini, B. L. West, G. W. Blake, and T. F. Wenisch. Q-zilla: A scheduling framework and core microarchitecture for tail-tolerant microservices. In *International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2020.

[72] B. Montazeri, Y. Li, M. Alizadeh, and J. Ousterhout. Homa: A Receiver-Driven Low-Latency Transport Protocol Using Network Priorities. In *Special Interest Group on Data Communication (SIGCOMM)*. ACM, 2018.

[73] J. Ousterhout. A linux kernel implementation of the homa transport protocol. In *Annual Technical Conference (ATC)*. USENIX, 2021.

[74] H. Qin, Q. Li, J. Speiser, P. Kraft, and J. Ousterhout. Arachne: Core-aware thread management. In *Operating Systems Design and Implementation (OSDI)*. USENIX, 2018.

[75] R. Ricci, E. Eide, and The CloudLab Team. Introducing CloudLab: Scientific infrastructure for advancing cloud architectures and applications. *USENIX ;login:*, 2014.

[76] L. Rizzo. Netmap: a novel framework for fast packet i/o. In *Security Symposium (USENIX Security)*. USENIX, 2012.

[77] A. Roy, H. Zeng, J. Bagga, G. Porter, and A. C. Snoeren. Inside the Social Network's (Datacenter) Network. In *Special Interest Group on Data Communication (SIGCOMM)*. ACM, 2015.

[78] R. Russell. virtio: towards a de-facto standard for virtual I/O devices. In *SIGOPS Operating Systems Review*, volume 42, pages 95–103. ACM, 2008.

[79] A. Saeed, N. Dukkipati, V. Valancius, V. The Lam, C. Contavalli, and A. Vahdat. Carousel: Scalable traffic shaping at end hosts. In *Special Interest Group on Data Communication (SIGCOMM)*. ACM, 2017.

[80] A. Saeed, Y. Zhao, N. Dukkipati, E. Zegura, M. Ammar, K. Harras, and A. Vahdat. Eiffel: Efficient and flexible software packet scheduling. In *Networked Systems Design and Implementation (NSDI)*. USENIX, 2019.

[81] Y. Shan, Y. Huang, Y. Chen, and Y. Zhang. Legoos: A disseminated, distributed OS for hardware resource disaggregation. In *Operating Systems Design and Implementation (OSDI)*. USENIX, 2018.

[82] E. Sharafzadeh, A. Sanaee, E. Asyabi, and M. Sharifi. Yawn: A cpu idle-state governor for datacenter applications. In *SIGOPS Asia-Pacific Workshop on Systems (APSys)*. ACM, 2019.

[83] A. Singhvi, A. Akella, D. Gibson, T. F. Wenisch, M. Wong-Chan, S. Clark, M. M. K. Martin, M. McLaren, P. Chandra, R. Cauble, H. M. G. Wassel, B. Montazeri, S. L. Sabato, J. Scherpelz, and A. Vahdat. 1RMA: Re-envisioning remote memory access for multi-tenant datacenters. In *Special Interest Group on Data Communication (SIGCOMM)*. ACM, 2020.

[84] A. Sivaraman, S. Subramanian, M. Alizadeh, S. Chole, S.-T. Chuang, A. Agrawal, H. Balakrishnan, T. Edsall, S. Katti, and N. McKeown. Programmable packet scheduling at line rate. In *Special Interest Group on Data Communication (SIGCOMM)*. ACM, 2016.

[85] A. Sivaraman, S. Subramanian, M. Alizadeh, S. Chole, S.-T. Chuang, A. Agrawal, H. Balakrishnan, T. Edsall, S. Katti, and N. McKeown. Programmable packet scheduling at line rate. In *Special Interest Group on Data Communication (SIGCOMM)*. ACM, 2016.

[86] A. Sriraman and A. Dhanotia. Accelerometer: Understanding acceleration opportunities for data center overheads at hyperscale. In *Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. ACM, 2020.

[87] B. Stephens, A. Akella, and M. Swift. Loom: Flexible and efficient NIC packet scheduling. In *Networked Systems Design and Implementation (NSDI)*. USENIX, 2019.

[88] B. Stephens, A. L. Cox, A. Singla, J. Carter, C. Dixon, and W. Felter. Practical DCB for improved data center networks. In *Conference on Computer Communications (INFOCOM)*. IEEE, 2014.

[89] A. Tootoonchian, A. Panda, C. Lan, M. Walls, K. Argyraki, S. Ratnasamy, and S. Shenker. ResQ: Enabling SLOs in network function virtualization. In *Networked Systems Design and Implementation (NSDI)*. USENIX, 2018.

[90] W. Tu, Y.-H. Wei, G. Antichi, and B. Pfaff. Revisiting the open vswitch dataplane ten years later. In *Special Interest Group on Data Communication (SIGCOMM)*. ACM, 2021.

[91] V. Vasudevan, A. Phanishayee, H. Shah, E. Krevat, D. G. Andersen, G. R. Ganger, G. A. Gibson, and B. Mueller. Safe and effective fine-grained TCP retransmissions for datacenter communication. In *Special Interest Group on Data Communication (SIGCOMM)*. ACM, 2009.

[92] J. Yang, Y. Yue, and K. V. Rashmi. A large scale analysis of hundreds of in-memory cache clusters at Twitter. In *Operating Systems Design and Implementation (OSDI)*. USENIX, 2020.

[93] T. Yu, S. A. Noghabi, S. Raindel, H. Liu, J. Padhye, and V. Sekar. FreeFlow: High Performance Container Networking. In *Workshop on Hot Topics in Networks (HotNets)*. ACM, 2016.

[94] Z. Yu, C. Hu, J. Wu, X. Sun, V. Braverman, M. Chowdhury, Z. Liu, and X. Jin. Programmable packet scheduling with a single queue. In *Special Interest Group on Data Communication (SIGCOMM)*. ACM, 2021.

[95] Y. Yuan, M. Alian, Y. Wang, R. Wang, I. Kurakin, C. Tai, and N. S. Kim. Don't forget the I/O when allocating your LLC. In *International Symposium on Computer Architecture (ISCA)*. ACM, 2021.

[96] D. Zats, T. Das, P. Mohan, D. Borthakur, and R. Katz. DeTail: Reducing the flow completion time tail in datacenter networks. In *Special Interest Group on Data Communication (SIGCOMM)*. ACM, 2012.

[97] X. Zhan, R. Azimi, S. Kanev, D. Brooks, and S. Reda. Carb: A c-state power management arbiter for latency-critical workloads. In *Computer Architecture Letters*, volume 16, pages 6–9. IEEE, 2016.

[98] Y. Zhang, Y. Tan, B. Stephens, and M. Chowdhury. Justitia: Software multi-tenancy in hardware kernel-bypass networks. In *Networked Systems Design and Implementation (NSDI)*. USENIX, 2022.

[99] Y. Zhu, H. Eran, D. Firestone, C. Guo, M. Lipshteyn, Y. Liron, J. Padhye, S. Raindel, M. H. Yahia, and M. Zhang. Congestion control for large-scale RDMA deployments. In *Special Interest Group on Data Communication (SIGCOMM)*. ACM, 2015.

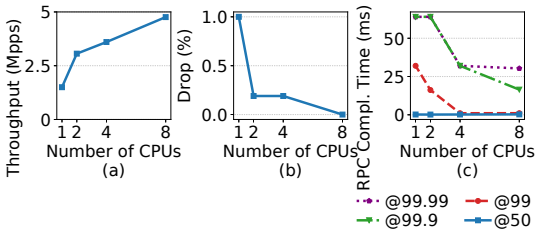


Figure 17: Throughput, loss and latency of DCTCP given different number of allocated cores. DCTCP still is unable to prevent packet loss with 4 CPU cores.

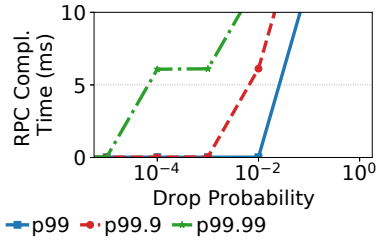


Figure 18: Homa experiences millisecond scale tail latency with even 10^{-2} drop probability.

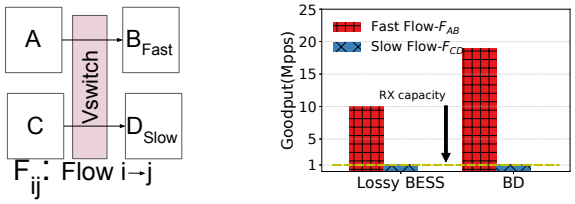


Figure 19: Backdraft TX bandwidth management in presence of a slow receiver. (a) The experiment setup on a single machine. (b) Backdraft prevents packet loss and saves CPU utilization from slow receiver and can allocate it to other receiver applications.

A Appendices

In this section, we expand our experiments associated with congestion controls, and bandwidth management on a single host.

A.1 Slow Receivers and DCTCP/Homa

While we discussed the problem associated with congestion controls such as Homa with regard to the slow receiver problem in Figure 4, we extended our study and performed similar experiments on DCTCP (§A.1.1).

We then discuss the impact of packet loss on latency of Homa (§A.1.2), given that Homa uses high granular timers to identify lost packets which is already discussed in §6.1.

A.1.1 DCTCP

We show that congestion control algorithms fail to address slow receiver problem in §2.1. Other than Homa, we performed the same test on DCTCP congestion control. Figure 17a show that throughput of DCTCP application cannot reach higher than 5 Mpps or 320 Mbps with even 8 cores (64 B packets were used).

We have found that this packet loss occurs even when the vswitch performs ECN marking and end-hosts use a state-of-the-art congestion control algorithm like DCTCP [12]. This is demonstrated in Figure 17b, which shows what happens when we vary the number of allocated cores from 1 to 8 allocated to a DCTCP receiver application experiencing receiving data from a DCTCP client that is utilizing 8 CPU cores to send messages as fast as possible. We enable ECN marking at *vswitch level* to ensure DCTCP controls the flow rates in the scenarios where only vswitches are involved. Finally, Figure 17c demonstrates that packet loss has dramatic impact on the tail latency of the DCTCP.

A.1.2 Packet Loss Effect on Homa

In this section, we further discuss packet loss overhead of Homa protocol discussed in §4. In Figure 18, we observe that RPC completion time increase to 5x higher with mere packet loss probability of 10^{-2}. Although Homa identifies lost packets with high resolution timers, this does not seem to be highly effective.

A.2 Single Host Bandwidth Management

We performed an extra experiment to show how Backdraft works when dealing with a non-cooperative workload in terms of bandwidth management. This experiment is carried on a single node, we demonstrate that Backdraft delivers 2x higher throughput than its counterpart, BESS. Figure 19a shows the setup for this experiment. Here we have four applications (A, B, C, and D), where application D is a slow receiver and process packets at a maximum of 1 Mpps. The sender applications (i.e., A and C) are configured to transmit packets at 20 Mpps, instead. Reciver B is not limited in performance, so we can consider it to be fast. When Backdraft identifies the queue buildup due to slow receiver (i.e., D), it sends a local overlay message towards the sender port that includes a pause duration and an estimate of the receiver's rate. Using this information, Backdraft can pause the sender port, save CPU cycles otherwise wasted in handling the slow receiver flow, and use the saved resources to better handle the traffic directed to the fast receiver.

Figure 19b demonstrates this. With Backdraft, flow f_{AB} achieves 19 Mpps throughput. BESS, however, wastes CPU cycles and throughput bandwidth on dropping packets, causing the flow to reach only 10 Mpps.