Opportunistic Packet Forwarding for Proactive Transport in Datacenters

Amir Shani*, Sogand Sadrhaghighi*, Mahdi Dolati† and Majid Ghaderi*
*Department of Computer Science, University of Calgary, Calgary, Canada.
†School of Computer Science, Institute for Research in Fundamental Sciences (IPM), Tehran, Iran.
Emails: {amir.shani, sogand.sadrhaghighi, mghaderi}@ucalgary.ca, m.dolati@ipm.ir

***Abstract*—Proactive transport protocols in datacenters are designed to avoid congestion by regulating flow sending rates via credit allocation. However, when a new flow starts, it takes one RTT before credits can be assigned to the new flow. To avoid stalling flows, modern proactive protocols such as NDP and Homa allow a new flow to blast a burst of unscheduled packets at line rate during the pre-credit phase. However, sending too many unscheduled packets could cause temporary traffic spikes that lead to queue build-ups, packet losses, and retransmissions, which are particularly detrimental to short flows. In this paper, we present the design and evaluation of *Opportunistic Packet Forwarding* (OPF), a data-plane building block for proactive transports designed to minimize pre-credit packet losses with negligible overhead on network switches. The key idea in OPF is to allow pre-credit packets to opportunistically take detours to avoid congested links on the shortest paths, effectively trading off packet losses for slightly increased delay. We have implemented OPF using P4 switches and integrated our implementation with both Homa and NDP. Our results on a range of traffic loads show significant improvement in the 99-th percentile of flow completion time for short flows, namely up to 54% reduction in NDP and 50% in Homa.**

I. Introduction

Thanks to advances in network hardware, modern datacenters have the potential to offer low-latency communication to distributed applications running in the datacenter. In fact, base round-trip-times (RTTs) of 5 μ-seconds or less are now prevalent in datacenters [27]. However, for applications that require fast data transfers within the datacenter, such short RTTs may not translate to proportionally faster completion times. The reason is that, the completion times of data transfers depend on both the high-speed network hardware and the ability of the network software running on the end hosts to take full advantage of the capabilities of the hardware.

Specifically, the sending rate of a traffic flow[1], and thus its completion time, is determined by a transport protocol such as TCP or its many datacenter optimized variants (*e.g.*, DCTCP [6] and DCQCN [38]). To determine the right sending rate for each flow in the network, the transport protocol continuously probes for available bandwidth, but reduces its sending rate if congestion is detected in the network. The consequences of using such protocols are that, not only it takes a few RTTs to converge to the right rate, but also they react to congestion only after the fact. This *reactive* behavior makes them ill-suited to modern datacenters in which the majority of flows are short [7]. While long flows are less sensitive to such behavior, short flows could be significantly impacted. Specifically, long flows requiring good average performance over many RTTs can tolerate transient sub-optimal sending rates over a few RTTs while the transport protocol tries to converge. But a few RTTs with sub-optimal performance could lead to unnecessarily long *tail latency* for short flows that could have been finished in just a few RTTs.

The latency of short flows has become critical due to several converging trends. First, the adoption of communication models such as remote procedure call (RPC) that consist of short requests and responses is making datacenter flows smaller in size. Second, as the link speeds increase, more flows can finish within a few RTTs. Indeed, measurements of production workloads show that more than 60% of flows can potentially finish in one RTT [7]. Third, the rate of increase in link speeds is outpacing the rate of increase in switch buffer spaces in datacenters [9], resulting in shallower buffers that make traffic bursts more likely to cause packet drops. Dropping even a single packet from a short flow can significantly increase its completion time. For example, for a short flow that could finish in one RTT, even if the most aggressive setting for the retransmission timeout is considered (*i.e.*, one RTT), the flow completion time will be inflated by $2\times$ (from 1 RTT to 2 RTT). Thus, to reduce the tail latency for short flows, it is critical to prevent packet drops and subsequent retransmissions.

Recently, an alternative approach to reactive transport protocols in datacenters, called *proactive* transport, has been considered [11], [20], [23], [27], [28] in which the right sending rate is proactively allocated to each flow either by the receiver or a central controller. The rate allocation is based on *credit* scheduling, where a flow is allowed to send a packet only if it has obtained a credit to do so. A straightforward implementation of this approach, however, still requires one RTT before credits can be assigned to a flow that is just starting. During this initial RTT, which is referred to as the *pre-credit phase* [23], no data can be sent even though the network could be lightly utilized. In particular, for a short flow, this could essentially double its completion time.

To avoid this problem, protocols such as NDP [20] and Homa [27] allow a new flow to blast a burst of packets, called *unscheduled packets*, at line rate during the first RTT. This allows short flows to potentially finish in the first RTT, while longer flows are regulated during the subsequent RTTs. The

[1] We use the term ‘flow’ to refer to a traditional 5-tuple IP flow.

ISBN 978-3-903176-63-8 © 2024 IFIP

downside of this approach is that sending many unscheduled packets could cause temporary traffic spikes that lead to queue buildups and eventually packet losses for both scheduled and unscheduled packets (even when a higher priority is given to scheduled packets [23]). In particular, datacenter workloads often require sending requests to a large number of worker processes and then handling their near-simultaneous responses, causing a problem known as *incast*. Such a communication pattern results in many short flows blasting unscheduled packets at a high rate in their pre-credit phase, which could create bursts that result in packet drops. As mentioned earlier, packet drops, even on a short time-scale, are catastrophic to short flows. As an example, Homa suggests setting its retransmission timeout to a few milli-seconds. For a flow that could be finished in 12 μ-seconds, which is common with 100 Gbps link speeds, a delay of 1.2 milli-seconds could mean a 100× increase in its flow completion time.

In summary, while proactive transport protocols such as Homa and NDP can substantially decrease the completion time of flows that last for multiple RTTs, they are still sub-optimal for tiny flows that are dominant in datacenter networks. Moreover, packet drops in the pre-credit phase are the main culprit for their sub-optimal performance. This is the problem we try to address in this work, namely *how to minimize packet drops during the pre-credit phase*, by proposing Opportunistic Packet Forwarding (OPF), a transport-agnostic solution to the pre-credit phase problem. Compared to a recently proposed pre-credit solution called Aeolus [23], OPF takes a fundamentally different approach. The approach proposed in Aeolus is to prioritize scheduled packets, selectively drop unscheduled ones, and then utilize fast retransmissions for loss recovery in the pre-credit phase. But even fast retransmissions take at least one RTT, inflating short flow completion times by at least 2×.

Our key insight in designing OPF is that due to the unbalanced load distribution in datacenters, there exists a distributed pool of under-utilized switch buffers in the network that have low occupancy during burst incidents. Additionally, due to high path redundancy, these under-utilized switch buffers can be exploited as detours for packets facing congestion, to avoid congestion with little increase in their end-to-end latency. This is exactly what OPF is designed for, namely *to divert congestion-facing unscheduled packets to under-utilized switch buffers*, thereby preventing catastrophic packet drops during the pre-credit phase. Other detour-induced buffer sharing solutions such as DIBS [35], detour packets indiscriminately during congestion, and are consequently prone to congestion spreading, infinite loop formation, and interfering with credit allocation logic of proactive transports which can result in extreme latency penalties [23]. Leveraging the P4 technology, OPF takes measures to limit network resource over-utilization, effectively controlling congestion spreading. Moreover, OPF can detour packets selectively and therefore, protect sensitive background traffic, *e.g.*, TCP, from packet reordering, as well as avoid interfering with the rate control and flow scheduling of proactive transports during the credit phase, by only detouring unscheduled packets.

The aim of OPF is to realize the original zero-loss goal of proactive transports without any modifications to the underlying transport protocols, or penalizing long flows. We show that OPF can be implemented at line rate with little overhead using commodity programmable switches. To this end, we have implemented OPF on software programmable switches with a P4 program, integrated it with two popular proactive transport protocols (*i.e.*, Homa and NDP) and conducted extensive experiments to analyze its impact on flow completion time (FCT) [2] for short flows and goodput for long flows.

Our contributions in this paper are:

- We present the design and evaluation of OPF, a light-weight solution to minimize packet drops in the pre-credit phase of proactive transport protocols. OPF requires only minimal modifications to the host stack and switch data-plane logic, and can be implemented on commodity programmable switches.
- As a proof-of-concept, we have implemented OPF in Mininet using P4 programmable software switches and Homa's Linux kernel module [29]. In our experiments with a 14-node network setup, we observed that OPF cuts the p99[3] of Homa's FCT by 60%. We also experiment with UDP workloads to demonstrate that OPF can be added to other transport protocols (such as those developed for UDP-based RoCEv2 [8]) as a generic technique for improving FCT of short flows.
- We have incorporated OPF in the same simulation software (using the same code base) that was used to evaluate Homa and NDP, namely `OMNeT++` [3] for Homa and `htsim` [2] for NDP. Our results show that Homa+OPF leads to 1.5× reduction in the median FCT and over 50% reduction in the p99 FCT compared to Homa. Compared to NDP, we observe that NDP+OPF reduces the median FCT by up to 6.8× and the p99 FCT by up to 54%.

The rest of the paper is organized as follows. The design of OPF is presented in Section II. Implementation details are covered in Section III. Evaluation methodology as well as Mininet and simulation results are presented in Section IV. Related works are reviewed in Section V, while Section VI concludes the paper and discusses potential future directions.

II. Design

OPF is designed to be simple and fast to mitigate packet drops by detouring them on the fly entirely in the data plane at line rate. In particular, OPF design is inspired by two well-known techniques: Random Packet Spraying (RPS) [14] and Random Early Detection (RED) [15]. OPF combines aspects of RPS and RED to achieve a low-latency and light-weight solution in absorbing micro-bursts during transient congestion events. Unlike RPS, OPF is congestion-aware and acts only when congestion is detected. Similar to RED, OPF is a switch-local solution and does not keep per-connection states at

[2] Unless otherwise specified, in this paper, the term 'FCT' is used to refer to the flow completion time for short flows.

[3] We use the term 'p99' to denote the 99-th percentile.

the switch. However, OPF acts in much smaller time scales compared to RED and only controls transient queue buildups.

A. Design Motivation

The design of OPF is motivated by the following well-known observations about datacenters:

High Path Redundancy. Datacenters are structured in a way to provide scalable and multi-path connectivity between switches using hierarchical Clos topologies [4], [18]. A consequence of this design is that there are multiple paths with similar RTTs between any pair of hosts in a datacenter [14]. In fact, path redundancy and symmetry have been exploited before for handling link failures [4], [5], [25], and load balancing [14] in datacenters.

Unbalanced Load Distribution. Datacenter switches rarely experience balanced utilization among their ports in the short term [36]. Also, a significant amount of traffic never leaves the server rack [10]. Therefore, datacenter traffic is unbalanced within switches as well as across topology layers, with the primary cause of packet drops being transient traffic spikes (*i.e.*, micro-bursts) in the network [10], as opposed to consistent over-utilization of links. In fact, the average link utilization in datacenters is relatively low due to over-provisioning.

B. Design Requirements

In order to have a practical solution, we have designed OPF such that the following requirements are satisfied:

Seamless Integration. OPF is designed to work with commodity programmable switches with minimal modifications to the host stack or switch software. Specifically, OPF is designed as a stand-alone building block that does not make any particular assumption about the underlying transport protocol, except for being able to tolerate benign out-of-order packets [14].

Packet-Scale Reaction. OPF reacts at line rate to congestion by being implemented entirely in the data plane. Any mechanism that relies on the control plane or software implementation introduces high latency, and as such is not suitable for dealing with packet drops in the pre-credit phase.

Stateless Operation. Due to memory limitations at the data plane [37], OPF keeps as little state as possible. In particular, OPF only keeps track of egress queue lengths on the switch, so that queue length information is available at ingress. However, it does not keep any states regarding the flows.

C. OPF Operation

On the host, all OPF does is marking unscheduled packets during the pre-credit phase of flows so that they are distinguishable by switches from the scheduled ones. Upon the arrival of a packet at the ingress control block of a switch, OPF observes the queue length at the packet's destination port. If the queue length exceeds a congestion detection threshold, congestion is detected. In such cases, OPF detours the congestion-facing unscheduled packets by forwarding them to other egress ports on the switch. While detouring may add extra delay by forwarding packets over longer paths, the impact of the extra delay is negligible compared to the cost of packet drops and retransmissions, as we show through extensive experiments in Section IV. Finally, OPF limits the number of times a packet gets detoured and therefore, exploits under-utilized network resources while curbing over-utilization by preventing the detour paths from getting excessively long.

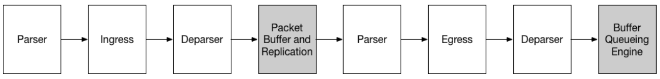


Fig. 1: High-level diagram of the PSA pipeline. [19]

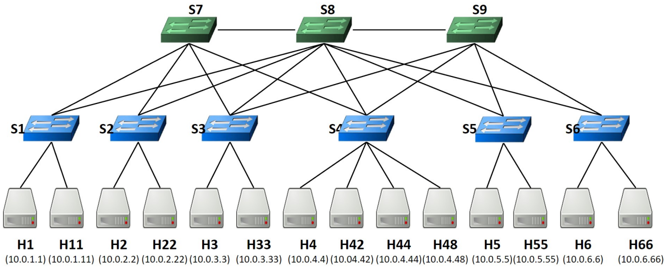


Fig. 2: The datacenter topology used in the Mininet setup. Each link has 100 Mbps bandwidth.

D. Design Elements

In the following, we present various design elements in OPF and discuss their operations.

Unscheduled Packet Marking. To avoid penalizing long flows, *e.g.*, by potentially creating larger queue backlogs as a consequence of not dropping packets, OPF only detours unscheduled congestion-facing packets. To do so efficiently, OPF marks unscheduled packets on the host by setting a bit in their headers. This way, every switch is able to parse the packet headers and distinguish the unscheduled packets from the scheduled ones without tracking the state of the flows.

Detour Count Check. In case of pervasive congestion throughout the network, the detoured packets could face congestion on multiple links along their path, and as such be detoured several times. While these additional detours increase the packet path length, OPF limits the path length increase of packets using a threshold on detour count denoted by `dcc-thresh`. Specifically, the number of times each packet has been detoured by OPF, henceforth referred to by DCC, is inferred from the TTL field of the IP header. To avoid congestion spreading, prevent network over-utilization, and limit the detour latency, OPF's switch module checks the value of DCC against `dcc-thresh` and does not detour packets with DCC higher than the threshold, allowing their fate to be decided by the rest of the packet processing pipeline.

Congestion Detection. There is a mapping between each egress port and an estimate of that port's queue length. When standard IP routing chooses an output port for the packet, OPF observes the queue length leading up to that port. To determine whether the port is congested, its queue length is compared against a threshold denoted by `opf-thresh`.

Port Selection. If the port is not congested, the packet is forwarded normally. But if the queue length surpasses the threshold, the packet has to be detoured in order to 1) avoid further increasing the congested port's queue length, and 2) prevent the packet from experiencing the queuing delay of that port or being dropped. The packet is detoured by selecting two ports at random and forwarding on the least congested one.

Queue State Update. The queue length information for each port is communicated from egress to ingress using global registers that map each egress port to its estimated queue length. This map is constantly updated in order to have an accurate estimate of egress queue length.

III. Implementation

We have implemented OPF on a P4 software switch to evaluate it through emulation. We used the bmv2 [13] software switch within a Mininet setup. The bmv2 interprets the output of a P4 compiler and emulates the packet-processing behavior specified by the compiled P4 program.

A. Host Implementation

OPF only requires a sending host to set the value of the OPF bit in each packet header. This bit, set at the host, indicates whether the packet belongs to the pre-credit phase of a flow. In our implementation, we have chosen the least significant bit of the ToS (Type of Service) field in the IP header to represent the OPF bit. Marking packets on the host is straightforward and can be done in several ways, *e.g.*, minimal modifications to the transport's host implementation, adding an extra marker logic using DPDK [1], or leveraging programmable smart NICs. In our implementation, this was achieved by modifying the Homa's Linux kernel module [29].

B. Switch Implementation

The bmv2 switch follows the PSA architecture [13]. P4 programs define the programming and interconnection of different programmable blocks within a target architecture, such as those that adhere to the PSA architecture [19]. Fig. 1 shows the high-level diagram of the PSA pipeline. When a packet enters the pipeline, it goes through multiple programmable blocks, each of which has a role in determining the packet's destination. In implementing OPF, the two main blocks on the path of the packet, namely *ingress* and *egress*, are used.

Ingress. In this block, a table lookup is performed based on the information in the packet's header. When a match is found, an action with proper parameters is invoked. In IP routing, the *output port* of the packet is one of the parameters which is chosen based on the routing decision. We call this parameter the *original egress port*. Once the IP forwarding action is invoked in the data plane, OPF intervenes as a hook. Specifically, OPF functions as a part of the packet processing pipeline and makes decisions based on the value of the OPF bit and the action parameters such as the original egress port. Algorithm 1 demonstrates the functionality of OPF within the ingress block. Specifically, OPF only processes the packet if OPF bit is set and DCC is less than the detour limit threshold `dcc-thresh`; otherwise, the packet proceeds to the rest of the pipeline (lines 3-4). The algorithm then checks for congestion at the original egress port (lines 6-8). If the corresponding queue length exceeds the threshold `opf-thresh`, then OPF detours the packet by forwarding it to the least congested of two randomly selected egress ports, effectively changing the packet forwarding outcome (line 13).

Algorithm 1: OPF Operation at Ingress.

```
1  port ← original egress port        /*From routing tables*/
2  DCC ← MAX_TTL - header.TTL         /*DCC estimate*/
3  if DCC ≥ dcc-thresh or header.OPF ≠ 1 then
4      goto 16
5  /* Get estimate of the original egress queue length*/
6  qlength ← qlenRegister[port]
7  /*Check for congestion*/
8  if qlength > opf-thresh then
9      /* Generate two random port numbers*/
10     randPort1, randPort2 ← random port numbers
11     /* Find the least congested one*/
12     detourPort ← $\arg\min_{x \in \{randPort1,\ randPort2\}} (qlenRegister[x])$
13     /* Set the egress port to the selected detour port*/
       std_metadata.egress_spec ← detourPort
14 else
15     /* Forward packet normally */
16     std_metadata.egress_spec ← port
```

Egress. After determining the output port and deciding on the forwarding decision in the ingress control block, the packet is forwarded to the queue of the chosen egress port and egress processing starts on the packet. The pseudo code for OPF's egress processing is shown in Algorithm 2. Specifically, OPF only updates the egress port queue length with the actual queue length of the port.

Algorithm 2: OPF Operation at Egress.

```
/*current port*/
index ← standard_metadata.egress_spec
qlengthRegister[index] ← queue length at the last
  packet dequeue
```

DCC Inference. The value of DCC can be inferred from the TTL field in the IP header, *i.e.*, it does not have to be explicitly encoded in the packet header. Specifically, we have

$$\text{DCC} \leq \texttt{MAX\_TTL} - \texttt{packet.TTL},$$

where `MAX_TTL` is the initial TTL value of IP packets in the network. This inequality sets an upper bound for DCC. Therefore, if OPF calculates (`MAX_TTL` − `packet.TTL`) and ensures it is less than `dcc-thresh`, then DCC is necessarily less than `dcc-thresh`.

Tracking Egress Queue. Due to the PSA design constraints, egress queue length information is not directly available at ingress [19]. To work around that, we have chosen to use registers available in P4 switches as persistent stateful memories. This also aligns with other works that rely on queue length feedback [26]. Registers can mediate information among control blocks, passing queue length information from egress to ingress ports [13].

IV. Evaluation

In this section, we first show the feasibility of OPF through emulation of its P4-based implementation. Then, we evaluate OPF with large-scale simulations using the simulators that were used in Homa [27] and NDP [21]. We investigate a diverse set of benchmarks with bursty incast combined with short tail and long tail background traffic. We test our hypothesis of whether integration of OPF with Homa and NDP can reduce the tail latency of short flows, without degrading the throughput of long flows. We provide insights into how OPF achieves latency improvements by analyzing the queue build-ups within switches IV-A and across topology layers IV-B. We use the following real-world workloads in emulation and simulations: (1) **Hadoop** from Facebook [31], and (2) **Web** from Microsoft [6]. `opf-thresh` is set to 4 to 12 packets lower than the maximum queue length in the experiments.

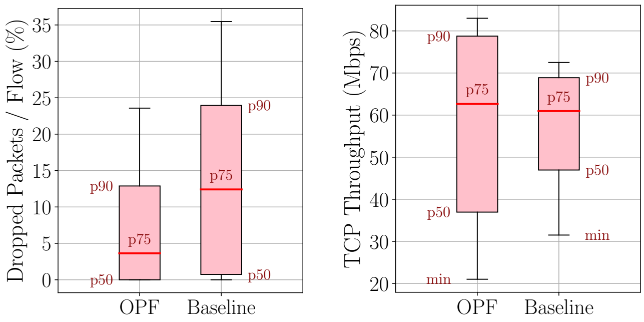
A. Proof of Concept Implementation

The objective of this subsection is to show the feasibility of implementing OPF on commodity programmable switches.

Setup. We set up a Mininet topology consisting of 14 hosts and 9 switches organized in a leaf-spine topology, as depicted in Fig. 2. All link bandwidths are set to 100 Mbps. Deploying our network emulation platform on a 192-core compute-optimized AWS instance provides ample resources to ensure that our emulation experiments yield results that closely mirror those of a physical deployment for a network of this size and practical traffic loads. We evaluate OPF using synthetic and real-world workloads. Our synthetic workload evaluates OPF using a blend of protracted TCP flows and bursts of UDP traffic. We then incorporate OPF with the Homa kernel module and test its effect on Hadoop and Web workloads.

Synthetic Traffic. We synthesized a combination of protracted TCP flows and bursty UDP traffic. OPF can use the protocol field in the IP header to distinguish these protocols. We then instructed OPF to ignore TCP traffic and detour UDP datagrams to showcase the effects of its detouring mechanism in the presence of background traffic, such as TCP, that is sensitive to packet reordering. We measured the throughput for TCP flows and packet drops for UDP flows under OPF and compared them to the case where switches have standard forwarding functionality.

Fig. 3 shows that OPF prevents excessive queue build-up and reduces the number of packet drops. Specifically, Fig. 3(a) compares the number of packet drops between OPF and the basic forwarding approach, *i.e.*, Baseline. TCP flows also benefit from the queue length control achieved by detouring UDP packets. This is illustrated in Fig. 3(b), which shows p90 and p75 improvements in TCP traffic throughput.

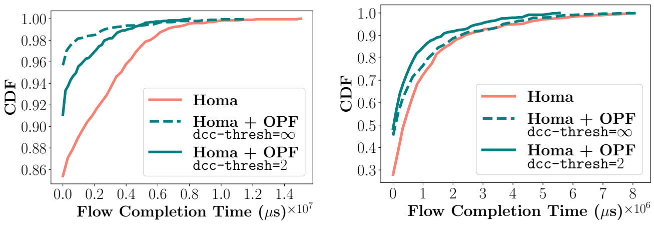


(a) Percentage of dropped UDP packets shows that OPF reduces packet drops through packet detouring.

(b) The increase in TCP median throughput and higher percentiles shows that TCP flows benefit from OPF.

Fig. 3: Mininet experiments comparing OPF and baseline forwarding under the synthetic UDP and TCP workloads.

Real-World Workloads. We conducted experiments using the Hadoop and Web workloads supplemented with incast traffic. In addition, we incorporated the Homa Linux kernel module into the hosts of our network to demonstrate the efficacy of OPF in proactive transports in a real-world scenario. Fig. 4 demonstrates the CDF of FCT when using Homa's kernel implementation. The figure provides a comparison of the latency improvements gained when Homa is enhanced by OPF with `dcc-thresh=2`, and when the DCC limiting mechanism is disabled with `dcc-thresh=` ∞. The figure demonstrates that FCT improves with OPF in both cases. In other words, the benefits of the reduction in packet drops outweighs the cost of increased path length with OPF. However, when OPF does not impose a limit on the number of times packets get detoured, path lengths might increase to a point where the trade-off does not favour tail latency, despite improvements at the head of the distribution. This demonstrates the efficacy of the `dcc-thresh` parameter in preventing congestion spreading and network resource over-utilization.

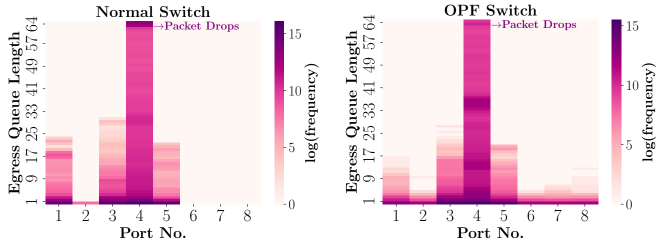


(a) CDF of FCTs under Hadoop workload.

(b) CDF of FCTs under Web workload.

Fig. 4: Mininet experiments with Homa's Linux kernel module showing the impact of OPF on Homa's FCT. When OPF limits the detour count for packets by setting `dcc-thresh=2`, tail latency improves significantly. Background traffic load is 0.25 in these experiments.

Queue Length Analysis. Analysis of switch queues is essential to delineate where gains of OPF come from. To this end, we investigated switch port states during traffic bursts in our network. Specifically, we measured the length of packet queues leading up to egress ports of the switches and recorded them in packet transmission-time granularity. Investigating the recorded queue length data, we observed that under the same bursty traffic pattern, the egress ports experience different queue length distributions using OPF compared to normal forwarding behaviour. The results are illustrated in Fig. 5 for one switch. Fig. 5, shows for all egress ports of the switch, how often each queue length was observed in the measurements, by attributing a heat color to each queue length value. Looking at Fig. 5(a), it is apparent that the bursts have caused port 4 to experience packet buffer saturation as indicated by the discernible heat value at the maximum queue length while there is little to no activity at four of the other ports. Fig. 5(b) shows how OPF mitigates the adverse effects of the bursts by shifting queue length concentration from maximum queue length to lower queue lengths at the congested port and elicits more balanced packet buffer utilization among other ports.



(a) The intense heat of the maximum queue length at port 4 indicates severe packet drop while buffer utilization is highly unbalanced among other ports.

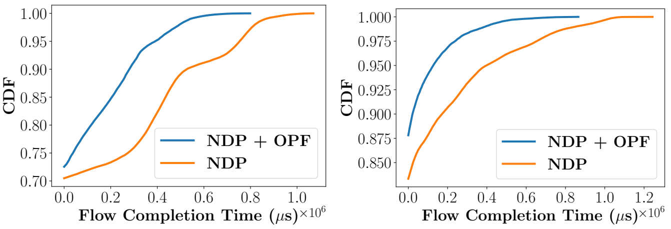
(b) OPF prevents packet drops as indicated by the reduced heat of maximum queue length at port 4, as well as more balanced activity among other ports.

Fig. 5: Queue length heat map for different ports of a switch experiencing bursts of packets. OPF reduces the concentration at the saturation queue length by diverting packets to other egress ports.

B. NDP-Based Large-Scale Simulations

NDP Integration. We have integrated OPF with NDP's simulator without replacing the core concepts of NDP. Flow sources in NDP start transmission by sending a number of packets specified by the *initial window* parameter at line-rate. Then, they wait for PULL packets from the receiver to adjust their sending rate. OPF marks a subset of the initial window packets as detour candidates. Recall that OPF never marks or detours scheduled packets. Moreover, OPF does not mark all the unscheduled packets. That is to limit the number of detouring candidates and avoid overflowing the spare queue capacity of the network. Therefore, the initial window size and the network's buffer capacity are the two parameters that bound the number of packets in each flow that OPF marks and subsequently detours.

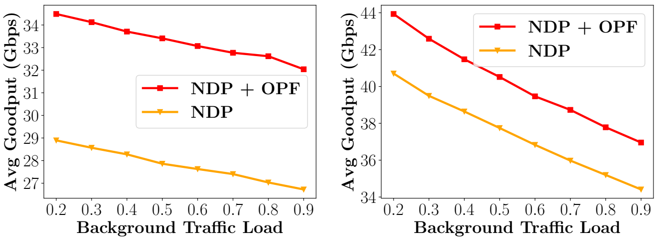
When a packet arrives at an overflowed switch port, the NDP switch, instead of dropping the packet, trims the packet and sends the packet header to the receiver. The receiver upon receiving a trimmed packet, generates a NACK and sends it to the sending host. The trimmed packet is inevitably retransmitted when the sending host receives the NACK, but this way, the sender is explicitly notified of the packet loss. OPF works as a hook in the packet processing pipeline of the NDP switch. When the queue length surpasses the `opf-thresh`, the hook activates and detours whole packets before the cut-payload mechanism of NDP trims the packets, in the hope of avoiding NACKs and ensuing retransmisisons.



(a) Hadoop Workload.

(b) Web Workload.

Fig. 6: FCTs of $0 - 100$ KB flows of different workloads under NDP and NDP+OPF. The traffic consists of incast scenarios on top of random background traffic. The background traffic load is 30%. Incasts arrive at regular intervals depending on the workload type.

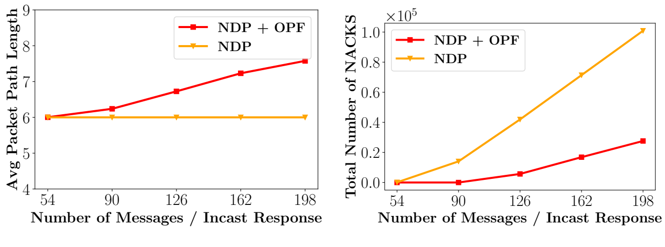


(a) Hadoop Workload.

(b) Web Workload.

Fig. 7: Average goodput of all flows of different workloads, consisting of incast and random background traffic, under NDP and NDP+OPF.

Setup. The network topology is the same 432-node FatTree used in the original NDP paper. The bandwidth of every link is set to 100 Gbps, and the output queue capacity of every switch port is set to 64 packets. To measure OPF's effect on NDP's FCT in the presence of bursts, we run experiments consisting of regular incast responses of various sizes destined to multiple servers in a destination pod. Many sources are generating these responses at different pods towards the destination pod receivers. All of this is in the presence of random background traffic between all the nodes at 30% load. The background traffic is the dominant traffic in terms of the number of flows. In terms of message size, the incast messages have the same size distribution as the background traffic.

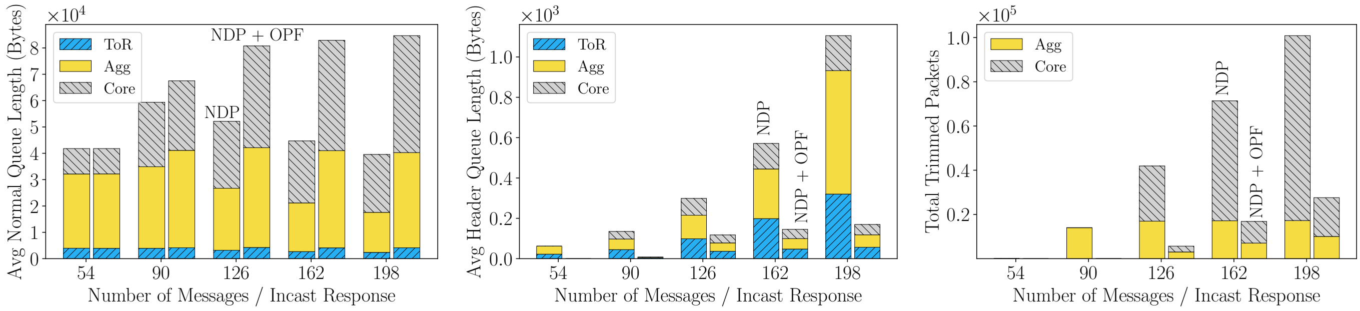


(a) Average path length of packets under NDP and NDP+OPF. OPF increases path lengths by detouring packets when they face congestion. NDP has constant path length.

(b) Total number of NACKs generated in network under NDP and NDP+OPF. The decrease in number of NACKs is because OPF prevents unnecessary cut-payloads.

Fig. 8: The trade-off between increasing packet path lengths by detouring them and decreasing costly retransmissions.

Results. In these experiments, we first demonstrate that OPF improves latency and goodput (Fig. 6 and 7). Then, to study the impact of OPF on the path and queue lengths in the



(a) NDP+OPF better utilizes the normal queues by preventing unnecessary cut-payloads at previous layers.

(b) NDP+OPF decreases header queue lengths.

(c) Comparison of cut-payloads happening at each topology layer between NDP and NDP+OPF.

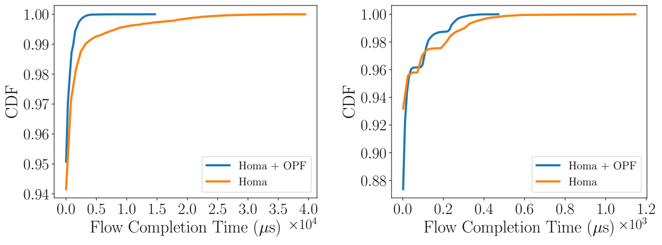
Fig. 9: Queue length comparison of NDP and NDP+OPF as the size of incast increases. For each incast size, the right-hand bar shows the data for NDP+OPF and the left-hand bar shows NDP's.

network, we design a specific scenario. In this scenario, we create incast without any background traffic (Fig. 8 and 9) where sources from different pods start sending short messages of 100 KB sizes to receivers in a destination pod in bursts. This scenario considers multiple bursts with a constant inter-burst arrival time.

- **Latency and Goodput.** Fig. 6 presents CDFs of the measured FCTs for the original NDP and OPF-enabled NDP referred to as NDP+OPF. The FCTs in the figure are for flows less than 100 KB in size. These results show $32-43\%$ improvement in p99 of FCT depending on the workload type. Moreover, we observe up to 4× median and 2× average FCT improvement. We also measure the average per-flow goodput for all the flows using the same experiment setup to verify that OPF does not degrade the throughput as a result of improved latency. Moreover, we vary the background traffic load to study the effect of traffic load on OPF's performance. The results in Fig. 7 demonstrate that OPF enhances the average goodput of flows by $7-38\%$ depending on the workload type. The key to understanding how OPF is able to improve both latency and goodput is to realize that OPF reduces the number of retransmissions while allowing for line-rate transmission of the latency-critical first-RTT portion of each flow.
- **Path Length Measurements.** Recall that OPF prevents data loss by detouring packets. There is an inherent trade-off here as OPF essentially avoids retransmissions at the cost of lengthening packet routes. Fig. 8 shows that OPF indeed increases path lengths (see Fig. 8(a)) but at the same time decreases the number of NACKs (see Fig. 8(b)) compared to NDP. We can observe that as the size of incast increases, NDP keeps the path length constant since the switches always adhere to the designated packet route at the source. Therefore, all packets facing congestion are trimmed and only the headers reach the destination, generating NACKs. This results in the drastically increased number of NACKs generated, with the increase in the incast size. OPF, on the other hand, keeps the number of NACKs lower by detouring packets and increasing their path lengths. An important note is that OPF still takes advantage of NDP's cut-payload mechanism while reducing the number of unnecessary NACKs.
- **Queue Length Measurements.** We have conducted fine-grained measurements of queues in the network to better demonstrate the buffer space utilization under NDP+OPF. Fig. 9(a) shows the size of the low priority queues, or "normal queues", that hold the full packets in the NDP switches at each layer of the topology. It is interesting to see that despite one's expectation, under NDP, the average size of normal queues across different topology layers at the destination pod does not have a direct relationship with the size of incast. It increases initially, but after some point starts shrinking. NDP+OPF, on the other hand, shows a different behaviour: queue sizes keep increasing until congestion reaches a point that saturates them. To identify the cause of this difference, we have sampled the high priority queues, or "header queues", that hold all the high priority packets including the header packets. The results are depicted in Fig. 9(b). These results, alongside the number of payload cuts at each layer (see Fig. 9(c)), clarify this behaviour. As incast intensifies, congestion starts propagating towards higher topology layers. Under NDP, the first layer that experiences congestion, starts cutting payloads and generating header packets (as demonstrated in Fig. 9(c). Fig. 9(b) corroborates this by showing a drastic increase in header queue size with the increase in incast intensity. As the congested layers start generating more header packets, they send them towards the receiver causing the next layers to see fewer full packets and more trimmed packets. This results in an under-utilization of queues at the next layer (as illustrated in Fig. 9(a)) since they have the capacity to forward full packets but instead, are receiving headers. OPF keeps more full packets in the network and makes use of the spare buffer capacity.

C. Homa-Based Large-Scale Simulations

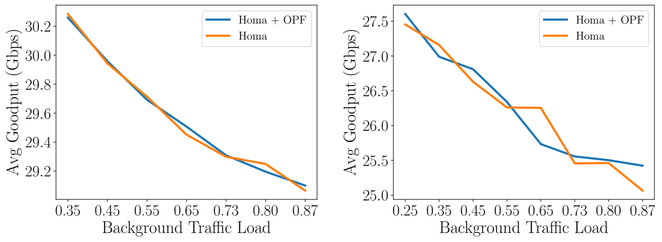
Homa Integration. We integrated OPF with Homa's OMNeT++ simulator. Our setup was the same as the original Homa paper with a 2-tier FatTree topology consisting of 144 hosts. We increased the speed of Host links from 10 Gbps to 40Gbps and the ToR-aggregation link speeds from 40Gbps to 100Gbps which are abundant in today's datacenter networks. The queue capacity of each egress switch port was set to 64 packets. We have reused and modified the retransmission mechanism added to Homa's OMNeT++ simulation by the Aeolus authors [23] optimizing it by adding random back-off



(a) Hadoop Workload.

(b) Web Workload.

Fig. 10: FCT of flows under Homa and OPF-enabled Homa. The traffic consists of incast scenarios on top of random background traffic. The background traffic load is $25\% - 35\%$. Incasts arrive at regular intervals depending on the type of the workload.



(a) Hadoop Workload.

(b) Web Workload.

Fig. 11: Average goodput measured for all the flows during experiments consisting of incast and random background traffic.

to the sender side timeout in order to prevent global synchronization in case of incast. Homa's retransmission timeout has been adjusted to micro-second values to align with the higher link speeds. The workloads were similar to those used in the evaluation of NDP, *i.e.*, incast responses of various sizes in combination with random background load with the same flow size distributions as those used with NDP.

Results. The FCT results for each workload are shown in Fig. 10. The experiments in Fig. 10 have moderate background load values of $25\% – 35\%$. The results show that Homa+OPF has $26\% - 50\%$ lower p99 tail latency compared to Homa. This is because OPF detours packets at the congested switch ports to the spare buffer capacity available in the network instead of dropping and subsequently retransmitting them after a timeout. As with the NDP experiments, we have measured the average per-flow goodput of Homa+OPF and compared it to Homa's while varying the background load. The results are presented in Fig. 11. We observe that Homa+OPF goodput performance is on par with Homa's. Hence, OPF has effectively improved the latency of short flows without incurring goodput degradation to longer flows.

V. Related Works

Credit-Based Transport. ExpressPass [12] is a credit-based transport protocol, where end-hosts distribute credit packets, while switches and NICs rate-limit the credit packets. ExpressPass uses explicit packets sent from the sender to the receiver to start and stop the flow of credit packets, which introduces one RTT delay. Another approach is proposed in pHost [16] where each flow is assigned a few free credits to avoid the one RTT delay required to receive its credit allocation. Similarly, senders in NDP [21] and Homa [27] start their transmissions before receiving credits, but their sending rate is regulated by an initial window size. Employing OPF would reduce the chance of dropping these unscheduled packets. Aeolus [23] takes a different approach by allowing flows to start at line-rate, but uses an in-network mechanism to selectively drop unscheduled packets to protect the scheduled packets. OPF is orthogonal to this technique and can reduce the number of dropped unscheduled packets under Aeolus.

Adaptive Routing. In adaptive routing, the pre-determined path of a packet can change during transmission in response to changes in the network. Various forms of adaptive routing have been proposed for HPC clusters [17], [24], [32], [33] to better utilize high-radix topologies such as the flattend butterfly or dragonfly in HPC. While the idea of avoiding congestion zones in the network is similar to OPF, there are major differences between the two approaches. Adaptive routing is a slow control plane technique that is more suitable for dealing with semi-persistent traffic spikes in order to amortize the cost of indirect routing, while OPF is a fast data plane forwarding technique (as opposed to routing) that has minimal overhead. Deploying adaptive routing requires changing the global routing structure of the network to support indirect routing via intermediate hosts, while OPF is independent of the specific routing mechanism deployed in the network.

Load Balancing. Oblivious load balancing techniques [22], [33], [34] can not completely eliminate network congestion or imbalanced link utilization. Load-aware end-to-end techniques [5], [25], [30] suffer from the same slow reaction problem as traditional reactive congestion control approaches. OPF works in the data plane; and while it is congestion-aware, it does not depend on end-to-end congestion information to reroute a packet, which allows it to operate over packet timescale (as opposed to RTT time-scale). Switch-local random forwarding techniques such as DIBS [35] and RPS [14] can be considered variants of OPF in theory. Specifically, by adjusting the `opf-thresh` and `dcc-thresh` parameters properly, OPF can operate as either DIBS or RPS. For example, by setting `dcc-thresh`$=\infty$, OPF can operate as DIBS during the pre-credit phase. However, the consequent unconstrained increase of path lengths might hinder tail latency improvements (section IV-A). In practice, DIBS and RPS both have hardware implementations that lack flexibility to accommodate the modern datacenter environments, whereas, OPF with its P4 implementation, can be easily integrated with commodity programmable switches, and deploy selective detouring mechanisms that make it more suitable for datacenters where different transports might coexist. For example, DIBS's nonselective packet detouring might interfere with TCP's fast retransmission mechanism by reordering TCP packets. Whereas, OPF can detect TCP packets by parsing packet headers at line rate and avoid detouring them (section IV-A).

VI. Conclusion

In this paper, we presented the design and evaluation of OPF, a light-weight data-plane mechanism to minimize the impact of traffic spikes in datacenters. The key idea in OPF is to exploit the under-utilized buffer space across network switches by opportunistically detouring congestion-facing packets in the network. We implemented OPF in software switches and Linux kernel. Our extensive emulation and simulation results show that integrating OPF with Homa and NDP leads to significant improvements in p99 completion time of short flows without negatively impacting the goodput of long flows. Our experiment results show that OPF's parameters have significant impact on OPF's performance improvements. The choice of these parameters for an optimal outcome depends on network characteristics such as topology and traffic load. In this work, the OPF parameters were set experimentally which is not guaranteed to result in the optimal outcome. As a worthwhile effort to extend this work, we believe it is possible to design a mechanism that dynamically tunes the values of `opf-thresh` and `dcc-thresh` parameters.

References

[1] DPDK: Data Plane Development Kit. https://www.dpdk.org/. Accessed September 20, 2023.

[2] htsim. https://github.com/kellianhunt/htsim. Accessed September 20, 2023.

[3] OMNeT++ Simulation of Homa. https://github.com/PlatformLab/HomaSimulation. Accessed September 20, 2023.

[4] Mohammad Al-Fares, Alexander Loukissas, and Amin Vahdat. A scalable, commodity datacenter network architecture. *ACM SIGCOMM computer communication review*, 38(4), 2008.

[5] Mohammad Alizadeh, Tom Edsall, Sarang Dharmapurikar, Ramanan Vaidyanathan, Kevin Chu, Andy Fingerhut, Vinh The Lam, Francis Matus, Rong Pan, Navindra Yadav, et al. CONGA: Distributed congestion-aware load balancing for datacenters. In *Proc. ACM SIGCOMM*, August 2014.

[6] Mohammad Alizadeh, Albert Greenberg, David A. Maltz, Jitendra Padhye, Parveen Patel, Balaji Prabhakar, Sudipta Sengupta, and Murari Sridharan. Datacenter TCP (DCTCP). In *Proc. ACM SIGCOMM*, August 2010.

[7] Serhat Arslan, Yuliang Li, Gautam Kumar, and Nandita Dukkipati. Bolt: Sub-RTT congestion control for ultra-low latency. In *Proc. USENIX NSDI*, April 2023.

[8] InfiniBand Trade Association et al. Infiniband architecture specification release 1.2.2. http://www.infinibandta.org. Accessed September 20, 2023.

[9] Wei Bai, Kai Chen, Shuihai Hu, Kun Tan, and Yongqiang Xiong. Congestion control for high-speed extremely shallow-buffered datacenter networks. In *Proc. ACM Asia-Pacific Workshop on Networking (APNet)*, August 2017.

[10] Theophilus Benson, Aditya Akella, and David A Maltz. Network traffic characteristics of datacenters in the wild. In *Proc. ACM IMC*, November 2010.

[11] Inho Cho, Keon Jang, and Dongsu Han. Credit-scheduled delay-bounded congestion control for datacenters. In *Proc. ACM SIGCOMM*, August 2017.

[12] Inho Cho, Keon Jang, and Dongsu Han. Credit-scheduled delay-bounded congestion control for datacenters. In *Proc. ACM SIGCOMM*, August 2017.

[13] P4 Language Consortium. bmv2: The reference p4 software switch. https://github.com/p4lang/behavioral-model, Accessed September 20, 2023.

[14] Advait Dixit, Pawan Prakash, Y. Charlie Hu, and Ramana Rao Kompella. On the impact of packet spraying in datacenter networks. In *Proc. IEEE INFOCOM*, April 2013.

[15] Sally Floyd and Van Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on networking*, 1(4), 1993.

[16] Peter X Gao, Akshay Narayan, Gautam Kumar, Rachit Agarwal, Sylvia Ratnasamy, and Scott Shenker. pHost: Distributed near-optimal datacenter transport over commodity network fabric. In *Proc. ACM CoNEXT*, December 2015.

[17] Dan Gibson, Hema Hariharan, Eric Lance, Moray McLaren, Behnam Montazeri, Arjun Singh, Stephen Wang, Hassan MG Wassel, Zhehua Wu, Sunghwan Yoo, et al. Aquila: A unified, low-latency fabric for datacenter networks. In *Proc. USENIX NSDI*, April 2022.

[18] Albert Greenberg, James R Hamilton, Navendu Jain, Srikanth Kandula, Changhoon Kim, Parantap Lahiri, David A Maltz, Parveen Patel, and Sudipta Sengupta. VL2: A scalable and flexible datacenter network. In *Proc. ACM SIGCOMM*, August 2009.

[19] The P4.org Architecture Working Group. P4 portable switch architecture. Technical report, 2022.

[20] Mark Handley et al. Re-architecting datacenter networks and stacks for low latency and high performance. In *Proc. ACM SIGCOMM*, August 2017.

[21] Mark Handley, Costin Raiciu, Alexandru Agache, Andrei Voinescu, Andrew W Moore, Gianni Antichi, and Marcin Wójcik. Re-architecting datacenter networks and stacks for low latency and high performance. In *Proc. ACM SIGCOMM*, August 2017.

[22] Christian Hopps. Analysis of an equal-cost multi-path algorithm. Technical report, 2000.

[23] Shuihai Hu, Wei Bai, Gaoxiong Zeng, Zilong Wang, Baochen Qiao, Kai Chen, Kun Tan, and Yi Wang. Aeolus: A building block for proactive transport in datacenters. In *Proc. ACM SIGCOMM*, July 2020.

[24] Nan Jiang, John Kim, and William J. Dally. Indirect adaptive routing on large scale interconnection networks. In *Proc. ACM Annual International Symposium on Computer Architecture (ISCA)*, June 2009.

[25] Naga Katta, Mukesh Hira, Changhoon Kim, Anirudh Sivaraman, and Jennifer Rexford. Hula: Scalable load balancing using programmable data planes. In *Proce. ACM SOSR*, March 2016.

[26] Qingkai Meng and Fengyuan Ren. Lightning: A practical building block for rdma transport control. In *Proc. IEEE/ACM IWQOS*, June 2021.

[27] Behnam Montazeri, Yilong Li, Mohammad Alizadeh, and John Ousterhout. Homa: A receiver-driven low-latency transport protocol using network priorities. In *Proc. ACM SIGCOMM*, August 2018.

[28] Kanthi Nagaraj, Dinesh Bharadia, Hongzi Mao, Sandeep Chinchali, Mohammad Alizadeh, and Sachin Katti. NUMFabric: Fast and flexible bandwidth allocation in datacenters. In *Proc. ACM SIGCOMM*, August 2016.

[29] John Ousterhout. A Linux kernel implementation of the homa transport protocol. In *Proc. USENIX ATC*, July 2021.

[30] Mubashir Adnan Qureshi, Yuchung Cheng, Qianwen Yin, Qiaobin Fu, Gautam Kumar, Masoud Moshref, Junhua Yan, Van Jacobson, David Wetherall, and Abdul Kabbani. PLB: Congestion signals are simple and effective for network load balancing. In *Proc. ACM SIGCOMM*, August 2022.

[31] Arjun Roy, Hongyi Zeng, Jasmeet Bagga, George Porter, and Alex C Snoeren. Inside the social network's (datacenter) network. In *Proc.ACM CSIGCOMM*, August 2015.

[32] A. Singh. *Load-Balanced Routing in Interconnection Networks*. PhD thesis, Stanford University, 2005.

[33] L. G. Valian. A scheme for fast parallelcommunication. *SIAM Journal on Computing*, 11(2), 1982.

[34] Erico Vanini, Rong Pan, Mohammad Alizadeh, Parvin Taheri, and Tom Edsall. Let it flow: Resilient asymmetric load balancing with flowlet switching. In *Proce. USENIX NSDI*, March 2017.

[35] Kyriakos Zarifis, Rui Miao, Matt Calder, Ethan Katz-Bassett, Minlan Yu, and Jitendra Padhye. Dibs: Just-in-time congestion mitigation for data centers. In *Proceedings of the Ninth European Conference on Computer Systems*, pages 1–14, 2014.

[36] Qiao Zhang, Vincent Liu, Hongyi Zeng, and Arvind Krishnamurthy. High-resolution measurement of datacenter microbursts. In *Proc. ACM IMC*, November 2017.

[37] Xiaoquan Zhang, Lin Cui, Kaimin Wei, Fung Po Tso, Yangyang Ji, and Weijia Jia. A survey on stateful data plane in software defined networks. *Computer Networks*, 184, 2021.

[38] Yibo Zhu et al. Congestion control for large-scale RDMA deployments. In *Proc. ACM SIGCOMM*, October 2015.