dcPIM: Near-Optimal Proactive Datacenter Transport

Qizhe Cai
Cornell University

Mina Tahmasbi Arashloo
University of Waterloo

Rachit Agarwal
Cornell University

ABSTRACT

Datacenter Parallel Iterative Matching (dcPIM) is a proactive datacenter transport design that simultaneously achieves near-optimal tail latency for short flows and near-optimal network utilization, without requiring any specialized network hardware.

dcPIM places its intellectual roots in the classical PIM protocol, variations of which are used in almost all switch fabrics. The key technical result in dcPIM is a new theoretical analysis of the PIM protocol for the datacenter context: we show that, unlike switch fabrics where PIM requires $\log(n)$ rounds of control plane messages (for an n-port switch fabric) to guarantee near-optimal network utilization, the datacenter context enables PIM to guarantee near-optimal utilization with constant number of rounds (independent of the number of hosts in the datacenter)! dcPIM design builds upon insights gained from this analysis, and extends the PIM design to overcome the unique challenges introduced by datacenter networks (much larger scales and round trip times when compared to switch fabrics). We demonstrate, both theoretically and empirically, the near-optimality of dcPIM performance.

CCS CONCEPTS

• **Networks** → **Transport protocols**; **Network protocol design**; **Data center networks**;

KEYWORDS

Datacenter networks; Proactive transport; Flow scheduling

ACM Reference Format:
Qizhe Cai, Mina Tahmasbi Arashloo, and Rachit Agarwal. 2022. dcPIM: Near-Optimal Proactive Datacenter Transport. In *ACM SIGCOMM 2022 Conference (SIGCOMM '22), August 22–26, 2022, Amsterdam, Netherlands.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3544216.3544235

1 INTRODUCTION

Modern datacenter networks bear a striking similarity to switch fabrics [3, 16, 20, 42]: both of these are organized around Clos-like topologies using low-port count switches and both of these experience similar workloads—incast (many input ports having packets for the same output port), all-to-all (each input port having packets for each output port), one-to-one, etc. Yet, scheduling mechanisms in state-of-the-art datacenter transport designs [7, 18, 22, 24, 34, 36] are significantly different from those used in switch fabrics. Bridging this gap has the potential for datacenter transport designs to benefit from decades of foundational work on switch scheduling that has led to near-optimal switch fabric designs [8, 30–32, 41].

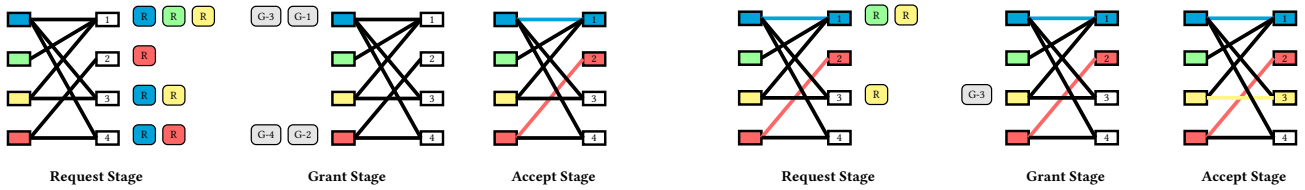
Datacenter Parallel Iterative Matching (dcPIM) is a proactive, receiver-driven, transport design that places its intellectual roots in classical switch scheduling protocols to simultaneously achieve near-optimal tail latency for short flows and near-optimal network utilization, without requiring any specialized network hardware. dcPIM achieves near-hardware latency for short flows by building upon ideas from recent transport protocols [7, 18, 22, 28, 34]. In particular, dcPIM is a connectionless protocol, allowing short flows to start sending at full rate; dcPIM uses per-packet multipath load balancing, that minimizes congestion within the network core; dcPIM enables lossless network only for control packets, allowing fast retransmission of lost short flow data packets in pathological traffic patterns (*e.g.*, extreme incast); and, dcPIM uses the limited number of priority queues in network switches to minimize interference between short flow and long flow data packets. We will show that, by carefully integrating these ideas into an end-to-end protocol, dcPIM achieves near theoretically optimal tail latency for short flows, even at 99^{th} percentiles.

What differentiates dcPIM from prior transport designs is that it achieves near-hardware tail latency for short flows while simultaneously sustaining near theoretically optimal (transient or persistent) network loads. dcPIM achieves this by placing its intellectual roots in the classical Parallel Iterative Matching (PIM) [8], a time-tested protocol variations of which are used in almost all switch fabrics. Just like PIM, hosts in dcPIM exchange multiple rounds of control plane messages to "match" senders with receivers to ensure that senders transmit long flow packets only when proactively admitted by the receivers. Realizing PIM-style matchings at the datacenter-scale turns out to be challenging: while PIM was designed for switch fabrics that have tens of ports and picosecond-scale round trip time (RTT), dcPIM needs to operate in a much harsher environment: datacenter networks have much larger scales and much larger RTTs. dcPIM resolves these challenges using two properties of datacenter environments. First, unlike switch fabrics where (at any point of time) each input port may very well have packets to send to each output port, it is rarely the case that (at any point of time) each host in the datacenter will have packets to send to each other host in the datacenter. That is, traffic matrices in datacenter networks are typically sparse: several studies from production datacenter networks show that, *when averaged across all hosts*, the number of flows at any point of time is a small constant [9, 17, 19, 37, 38]; furthermore, dcPIM performs matchings only for long flows that are likely to be even fewer in number. Second, unlike switch fabrics that are designed to run at full load, datacenter networks rarely run at an average load of 100%.

dcPIM leverages the first datacenter network property to establish a new theoretical result: unlike switch fabrics where PIM

SIGCOMM '22, August 22–26, 2022, Amsterdam, Netherlands

ACM ISBN 978-1-4503-9420-8/22/08…$15.00
https://doi.org/10.1145/3544216.3544235



(a) Round One (b) Round Two

Figure 1: An example of Parallel-Iterative Matching (PIM) (see §2 for a detailed discussion of the example.)

achieves near-optimal utilization using $\log(n)$ rounds of control messages (for an n-port switch fabric), traffic matrix sparsity in datacenter networks allows dcPIM to guarantee near-optimal utilization with constant number of rounds, that is, independent of the number of hosts in the network[1]. This result enables a dcPIM design that scales well, independent of the datacenter network size. dcPIM leverages the second property that datacenter networks rarely run at 100% load to *pipeline* data transmission between currently matched sender-receiver pairs with control messages for computing next set of matchings to hide the overheads of longer RTTs of datacenter networks.

While dcPIM builds upon a strong theoretical foundation, the final end-to-end design embraces simplicity just like the original PIM protocol: the number of matching rounds, the timescale for each matching round, and the number of data packets that can be sent upon matching are all fixed in advance. Just like PIM, dcPIM design is also robust to imperfection: it is okay for host clocks to be asynchronized—some of the control messages may be delayed within the fixed time used for matching rounds; the randomized matching algorithm in PIM combined with multiple rounds of control plane messages ensures that hosts unmatched in one round will be able to catch up in the remaining rounds (§3.1), and will continue to request matching until data transmission is complete (§3.2). The final result is a new proactive datacenter transport design that requires no specialized hardware, no per-flow state or rate calculations at switches, no centralized global scheduler, and no explicit network feedback and yet provides near-optimal performance both in terms of tail latency and network utilization.

We have implemented dcPIM in Linux hosts, and in simulations. dcPIM evaluation over a small-scale CloudLab testbed and over simulation demonstrates that dcPIM consistently achieves near-hardware latency and near-optimal network utilization across a variety of evaluation settings that mix-and-match three network topologies, three workloads, three traffic patterns, varying network topology oversubscription, and varying network loads. dcPIM simulator and implementation, along with all the documentation needed to reproduce our results, are available at `https://github.com/Terabit-Ethernet/dcPIM`.

[1]This result may be of independent interest. Getting back to our starting point for a moment—bridging the gap between scheduling mechanisms in switch fabrics and datacenter networks—our theoretical analysis reveals several interesting insights about existing proactive transport designs: (1) the superior performance of protocols like pHost and Homa on large-scale simulations is rooted in the sparsity of traffic matrices that they evaluate on; and (2) the poor network utilization of pHost, NDP and Homa (*e.g.*, as observed in Aeolus [24]) is due to traffic matrices that are *transiently* denser than what these protocols can handle. Our theoretical analysis establishes the sparsity of traffic matrices for which these protocols will observe poor network utilization (and our simulation results match the theoretical analysis)!

2 PARALLEL ITERATIVE MATCHING (PIM) OVERVIEW

Most modern switches use multi-stage Clos interconnects [42]. They can simultaneously deliver data from multiple input ports to multiple output ports, without any internal buffering and with each port operating at line rate, as long as there is one-to-one mapping between the input and the output ports: each input port sends data to at most one output port and each output port receives data from at most one input port. Thus, scheduling on the switch interconnect fabric reduces to finding a large number of conflict-free pairings of input and output ports with outstanding data, that is, the classic problem of finding matchings on bipartite graphs [25, 27, 29, 43]. PIM [8], and its variations, are one of the most widely deployed matching protocols in switch fabrics due to their simplicity, efficiency, and near-optimality (in terms of fabric capacity utilization); see §5. PIM uses multiple rounds of control messages to compute conflict-free pairings of input and output ports with outstanding data; each round consists of the following three stages:

- **Request Stage**: At the beginning of the round, each unmatched input port sends a `request` (control message) to each output port for which it has outstanding data.
- **Grant Stage**: Each unmatched output port picks one of the received `requests` uniform randomly, and sends a `grant` message to corresponding input port.
- **Accept Stage**: Each unmatched input port picks one of the received `grants` uniform randomly, and sends an `accept` message to corresponding output port. Both the input and the output ports mark themselves as matched.

An example. Consider the example in Figure 1. Output ports are numbered 1 to 4, and each input port has a unique color. In the first round (Figure 1(a)), all input ports are unmatched and will send requests to outputs ports for which they have outstanding data (shown as edges). For instance, the blue input port sends requests to output ports 1, 3, and 4. Next, each output port randomly selects a request to grant. In our example, output ports 1 and 3 send grants to the blue input port while the other two send grants to the red input port. Finally, each input port randomly picks a grant to accept and marks itself as matched. In our example, the blue input port accepts output port 1 and the red input port accepts output port 2. The red and blue input ports, and output ports 1 and 2 will mark themselves as matched (creating a matching of size 2) while the rest stay unmatched.

In the second round (Figure 1(b)), only the green and yellow input ports are unmatched, and thus, only output ports 1 and 3 receive requests. Out of the two, only output port 3 is unmatched. It sends a grant to the yellow input port, which the yellow input port

accepts. Thus, output port 3 and the yellow input port are matched, creating a matching of size 3. This is a maximal matching, that is, no more input-output port pairs will be added to the matching in subsequent rounds. Once the matching rounds are over, the matched input ports send data to their matched output ports.

PIM properties. PIM has several properties that have led to its widespread adoption in today's switch fabrics. First, it computes a near-optimal matching in $\sim \log n$ rounds, where n is the number of switch ports. Since switch fabrics have only tens of ports (small n), 6-8 rounds are enough to converge to a near-optimal matching.

Second, PIM embraces simplicity: the number of matching rounds, and the timescale of each stage and each round is fixed independent of the workload. For instance, fixing the number of rounds to 8 and fixing the timescale of each stage to 1.5× of the round trip time (RTT) between input and output ports (to account for queueing of `request`s and `grant`s as in Figure 1), the entire computation of matching takes 12 RTTs worth of time. In switch fabrics, the RTT between input and output ports is of the order of picoseconds; thus, matchings can be computed in a few picoseconds, a tiny fraction of the time it takes to transmit a packet. As a result, switch fabrics can simply execute matching and data transmission sequentially without harming latency or utilization. Note that randomized choices made at input and output ports, along with multiple rounds of matchings, allows PIM to be robust against the imperfection due to fixed timescales of each stage—delayed control packets are simply discarded since ports unmatched after a round get to participate in the matching protocol in subsequent rounds.

Finally, PIM computes matchings on per-packet granularity; in fact, to account for varying packet sizes, many switches schedule packets over the fabric at the granularity of fixed-sized cells. Once the input ports send one packet to their matched output ports, the matching is recomputed. Thus, it can effectively be used to maximize switch fabric utilization independent of the workload.

2.1 Challenges in realizing PIM for datacenters

Efficiently realizing PIM at the datacenter-scale requires resolving several challenges introduced by the datacenter environment. First, naïvely using PIM on datacenters can lead to high latency due to datacenter networks having much larger scales than switch fabrics: using $\sim \log n$ rounds on a datacenter network means that a flow may have to wait for tens of RTTs (which can typically amount to hundreds of microseconds) for its sender and receiver to match before it can be transmitted. To be able to efficiently realize PIM for datacenters, we need a better tradeoff between number of rounds and utilization guarantees.

Naïvely using PIM on datacenters can also lead to low network utilization. Even if we run PIM for fewer than $\sim \log n$ rounds, just computing matchings will still take tens of microseconds; thus, performing PIM-style sequential matching and data transmission (that is, compute a matching, transmit data between the matched hosts, and compute a new matching again) will result in network being idle during matching computations. This will waste tens of microseconds of data transmission time, harming utilization for modern high-bandwidth networks. In addition, computing matchings at per-packet granularity as in PIM could lead to even more underutilization since computing matchings will take much longer than packet transmission times. Thus, we need to perform matchings in a manner that the time taken to compute matchings perfectly matches the time taken to transmit the data upon matching.

Finally, unlike switch fabrics that typically operate in a failure-free manner and are designed to have full-bisection bandwidth, failures and oversubscription are a norm in datacenter networks. Thus, an efficient datacenter-scale realization of PIM requires additional mechanisms to handle failures and oversubscription.

3 dcPIM DESIGN

dcPIM design, as in the original PIM protocol, comprises of two phases: a matching phase and a data transmission phase. However, to overcome the aforementioned challenges of adapting PIM to datacenters, dcPIM design uses four key ideas. First, using a new theoretical analysis, dcPIM demonstrates that for the special case of datacenter networks, it is possible to achieve near-optimal network utilization with a small number of rounds in the matching phase (§3.1). Second, dcPIM uses a receiver-driven per-packet admission control mechanism to efficiently handle oversubscription and to enable fast data retransmission during failures (§3.2). Third, to hide the long datacenter RTTs, dcPIM carefully pipelines the matching and data transmission phases (§3.3). Finally, to ensure that the time taken to compute matchings perfectly matches the time taken to transmit the data between matched hosts, dcPIM extends PIM's original matching algorithm to match each sender with multiple receivers and vice versa (§3.4).

Several basic aspects of dcPIM design are borrowed from recent work on proactive receiver-driven transport designs [18, 34]. In particular, upon arrival of a new flow, the sender sends to the corresponding receiver a `notification` packet that may contain information for the receiver to make matching decisions (*e.g.*, flow size and deadline, tenant information, etc.), if such information is available. Switches perform randomized load balancing at a per-packet granularity (*e.g.*, using packet spraying [14]). All control packets in dcPIM are sent at the highest priority—as argued in [18, 22, 34], sending control packets at the highest priority, combined with buffer sizes in modern switches, ensures that loss of control packets is extremely rare in failure-free scenarios; dcPIM can thus use lightweight mechanisms to efficiently handle packet loss. Finally, for latency-sensitive short flows, the sender transmits the flow immediately with second highest priority; this mechanism, similar to [18, 22, 34][2], allows minimizing short flow completion times. For longer flows, dcPIM's mechanism kicks in: the flow is transmitted once the sender is matched with the corresponding receiver, and once the receiver admits packets for the flow. If additional priority levels are available, dcPIM uses the remaining priority levels for these flows, using a mechanism described in §3.2.

3.1 The Matching Phase

At its core, dcPIM matching phase is the same as PIM: senders and receivers exchange control messages over multiple rounds to match with each other using the protocol outlined in §2. A minor difference is that in dcPIM, receivers start the round by sending

[2]There is a subtle difference, though: while [18, 22, 34] transmit first few packets of *all* flows immediately, dcPIM does so only for short latency-sensitive flows. Network operators can configure which flows are considered short.

requests since they already get notified about, and keep track of, senders with outstanding data. The core difference between dcPIM and PIM is a new tradeoff between number of rounds and matching size based on a new theoretical analysis, which we outline next.

Exploiting traffic matrix sparsity for improved theoretical guarantees. Our analysis exploits the sparsity of traffic matrices in datacenter networks: several studies from production datacenters [9, 17, 19, 37, 38] show that the average incast/outcast degree in the network, or alternative the number of concurrent flows—*averaged over all senders/receivers*—is far less than the total possible number of flows (quadratic in number of hosts). For instance, [9] shows that, across seven different datacenters, the number of concurrent flows across a sample of 2000 servers is less than 10000 (that is, the *average* number of concurrent flows is less than 5); furthermore, recall that dcPIM performs matchings only for long flows; these are likely to even fewer in number at any given point of time. dcPIM design exploits such sparse traffic matrices. In particular, we prove the following theorem:

Theorem 1. *Let $\mathcal{G}$ be a connected bipartite graph with n nodes, and let $\bar{\delta}$ be the average degree. If PIM computes a matching of expected size $M^\star = n/\alpha$, for some $1 \le \alpha \le n$, using $O(\log n)$ rounds, then after r rounds, dcPIM computes a matching of expected size $(1 - \frac{\bar{\delta}\cdot\alpha}{4^r})M^\star$.*

This is a powerful result for two reasons. First, it shows that, for small constant values of $\bar{\delta}$, dcPIM computes a matching of roughly the same quality as PIM, but using only constant number of rounds—independent of the network size; thus, dcPIM's matching mechanism can scale independent of datacenter network size. For instance, in a one-million server datacenter with an *average* incast/outcast degree of 5, if 80% of the senders/receivers are matched by PIM, the above theorem guarantees that dcPIM will match $> 78\%$ of the senders/receivers using just 4 rounds of matching (unlike PIM, that will take $\approx \log n = 20$ rounds). The second reason the above result is powerful is because it explains why recent proactive transport protocols like pHost, NDP and Homa (that essentially do one round of matching) seem to perform so well over large-scale simulations: workloads over which these protocols are evaluated usually generate *extremely* sparse traffic matrices. The result also explains the recently observed poor throughput of NDP and Homa protocols [24]: when traffic matrices are transiently dense, single-round matching protocols like NDP and Homa result in extreme network underutilization since they are unable to sustain the load due to poor matching sizes; as we will show, our simulations confirm precisely this intuition and result.

To prove the above theorem, we have to employ a new proof strategy since merely replacing the total number of edges in PIM analysis ($= n^2$) by $\bar{\delta}\cdot n$ will not lead to a better bound ($O(\log(\bar{\delta}\cdot n))$ is asymptotically $O(\log n)$). In the original PIM proof, the core challenge was to prove the number of rounds in which the matching algorithm converges; once converged, the bound on matching size was quite trivial. On the other hand, we are already fixing the number of rounds, and the core challenge is to prove that the resulting matching size is close to optimal.

Proof of Theorem 1. Let us call a sender active if the sender is unmatched, and has at least one unmatched receiver to which it has an outstanding flow. A request (an edge in the graph) is said to be unresolved if both the sender and the receiver for the request are currently unmatched, and resolved otherwise.

Similar to the original analysis of PIM, we can show that after each matching round, the expected number of unresolved requests for a sender are 1/4 of the unresolved requests after the last round. Let $Q_{u,r}$ be a random variable representing the number of unresolved requests for sender u after r rounds. In round r, u receives q unresolved requests from different receivers. These receivers can be divided into two groups:

(1) Receivers that will not receive grants from any other senders. These receivers will be unmatched if u does not send a grant to them. Suppose there are k such receivers.
(2) Receivers that will receive at least one grant from other senders. These receivers will be matched anyway. There are $q-k$ receivers in this group.

If u sends a grant to a receiver in the first group, then u will definitely be matched. In that case, all of the q requests u has received in this round will be resolved in round r (note that, by definition, a request is resolved if its sender is matched even if the request itself is not granted). If u sends the grant to a receiver in the second group, u may not be matched, but receivers in group 2 will all be matched as they receive at least one other grant from other senders. Thus, in this case, at least $q-k$ requests becomes resolved. Combining these two, in expectation, at least $\frac{k}{q}\cdot q + \frac{q-k}{q}\cdot(q-k)$ requests will be resolved. Thus, the expected number of unresolved requests after r round at any sender u is at most $k\cdot\frac{q-k}{q}$ which is less than or equal to $\frac{q}{4}$ for any k when $q > 0$. That is, $\mathbb{E}[Q_{u,r}|Q_{u,r-1} = q] \le \frac{q}{4}$.

Using Bayes rule and linearity of expectation, we have:

$$\begin{aligned}\mathbb{E}[Q_{u,r}] &= \Sigma_{q=0}^{n} Pr(Q_{u,r-1} = q)\cdot\mathbb{E}[Q_{u,r}|Q_{u,r-1} = q]\\ &\le \Sigma_{q=0}^{n} Pr(Q_{u,r-1} = q)\cdot\frac{q}{4}\\ &= \frac{\mathbb{E}[Q_{u,r-1}]}{4}.\end{aligned}$$

Therefore:

$$\mathbb{E}[Q_{u,r}] \le \frac{\mathbb{E}[Q_{u,r-1}]}{4} \le \frac{\mathbb{E}[Q_{u,r-2}]}{4^2} \le \cdots \le \frac{\mathbb{E}[Q_{u,0}]}{4^r}.$$

Thus, after r rounds, the expected number of unresolved requests for a sender u is at most $\mathbb{E}[Q_{u,0}]/4^r$. The number of unresolved requests in round 0 (before we start) is fixed and equal to $\deg(u)$ (in the original PIM analysis, $\deg(u) = n$ for each sender). Therefore

$$\mathbb{E}[Q_{u,r}] \le \frac{\deg(u)}{4^r}.$$

From here on, our proof differs from PIM analysis. Let $I_{u,r}$ be an indicator random variable that is one if $Q_{u,r} \ge 1$ and zero otherwise. That is, $\mathbb{E}[I_{u,r}] = Pr(I_{u,r} = 1) = Pr(Q_{u,r} \ge 1)$. Markov inequality says for a nonnegative random variable X and $a > 0$, $Pr(X \ge a) \le \frac{\mathbb{E}[X]}{a}$. That is, the probability that X is at least a is at most the expectation of X divided by a. As a result, for the nonnegative random variable $Q_{u,r}$ and $a = 1$, we have

$$Pr(Q_{u,r} \ge 1) \le \mathbb{E}[Q_{u,r}] \le \deg(u)/4^r,$$

and therefore:

$$\mathbb{E}[I_{u,r}] = Pr(Q_{u,r} \ge 1) \le \deg(u)/4^r.$$

Let U be the set of all senders and let $\mathcal{A}$ be the set of senders that are active after r rounds. That is, $\mathcal{A}$ is the set of senders u with $Q_{u,r} \geq 1$. Intuitively, the rest of the proof builds upon following two observations. First, based on the above result, senders with degree less than 4^r will have their requests resolved after r rounds and will become inactive. Second, since senders in $\mathcal{A}$ have degree more than 4^r (since they are active), and since the average degree of the graph is bounded, the size of $\mathcal{A}$ cannot be very large. Specifically, the expected number of active senders after r rounds is

$$\mathbb{E}[|\mathcal{A}|] = \Sigma_{u \in U} \mathbb{E}[I_{u,r}] \leq \Sigma_{u \in U} \deg(u)/4^r \leq \bar{\delta} \cdot n/4^r$$

where $\bar{\delta}$ is the average degree. The expression above follows from the fact that, since the graph is bipartite, sum of the degrees of senders is equal to that of receivers (that is, $\Sigma_{u \in U} \deg(u) = \bar{\delta} \cdot n$).

If we let dcPIM run for additional rounds, in the best case, all senders in $\mathcal{A}$ will be added to the matching. Thus, the gap between matching computed by dcPIM after r rounds and the matching computed by PIM (after it converges in, say, $\log(n)$ rounds) is bounded by size of $\mathcal{A}$. Specifically, Let M_{PIM} be the size of the matching computed by PIM after $\log(n)$ rounds, and let $M^\star = \mathbb{E}[M_{PIM}] = n/\alpha$. Then, the size of matching computed by dcPIM after r rounds is:

$$\begin{aligned}
\mathbb{E}[M_{dcPIM}] &\geq M^\star - \mathbb{E}[|\mathcal{A}|] \\
&\geq \frac{n}{\alpha} - \frac{\bar{\delta} \cdot n}{4^r} \\
&= \frac{n}{\alpha} \cdot \left(1 - \frac{\bar{\delta} \cdot \alpha}{4^r}\right) \\
&= M^\star \cdot \left(1 - \frac{\bar{\delta} \cdot \alpha}{4^r}\right)
\end{aligned}$$

This completes the proof □

Matching size versus utilization guarantees. Our discussion so far has focused on size of the matching computed by dcPIM, and how this matching is near-optimal. However, once a receiver is matched to a sender, it admits packets only from that sender. Thus, matchings correspond to network utilization only if the sender has enough packets to be admitted by the receiver to "fill up" the entire data transmission phase. We will show, in §3.4, that by extending the matching algorithm to match each receiver with multiple senders and vice versa, dcPIM can ensure there are enough packets for the receivers to admit for almost the entire data transmission phase, hence providing network utilization guarantees same as the matching size guarantees.

3.2 The Data Transmission Phase

At a high-level, dcPIM data transmission phase uses a simple protocol: every MTU transmission time, the sender checks if it has a short flow for which packets can be transmitted without matching; if so, the sender transmits a packet from a short flow using the second highest priority. If the sender has no short flows, it must wait to match with the receiver before transmitting packets. Each receiver admits packets from the matched sender using a per-packet "token" that specifies the flow ID and the sequence number of the admitted packet. We provide details on the protocol below.

dcPIM uses four kinds of control packets in data transmission phase: `notification`, `finish`, `ack` and `tokens`; all control packets are transmitted at the highest priority. Upon a flow arrival, the sender sends a `notification` control packet to the corresponding receiver; the `notification` contains the flowID and may contain additional information about the flow (*e.g.*, flow size and deadline, tenant information, etc.), if available. Once the sender has finished transmitting all data packets for a flow, it sends a `finish` control packet to the corresponding receiver; the `finish` packet contains the flowID and information about the number of packets transmitted by the sender for that flow. Upon receiving a `notification`, the receiver immediately responds with an `ack` control packet; upon receiving a `finish` packet, the receiver immediately responds with a `finish` control packet if and only if it has received all the packets for that flow (which can be checked since sender's `finish` packet contains that information). Control packets being transmitted at the highest priority, combined with buffer sizes in modern switches, means that the datacenter network behaves like a lossless fabric for control packets in failure-free scenarios [18, 22]. To handle control packet drops (*e.g.*, due to failures), dcPIM uses a lightweight mechanism described in §3.5.

dcPIM token clocking mechanism for handling in-network congestion. Each receiver admits long flow packets from the matched sender at a per-packet granularity using tokens, where each token allows the sender to transmit one packet. To efficiently react to congestion in oversubscribed topologies, and to efficiently handle transient congestion due to high-priority short flows being transmitted without explicit matching, dcPIM receivers use a token clocking mechanism similar to TCP and pHost [18]. Each receiver maintains a per-flow sliding token window of a default size of 1 BDP, created upon receiving a `notification`. Tokens specify the flow ID, the admitted data packet sequence number, and cumulative acknowledgements (the smallest packet sequence number for which the token has been sent but the data packet has not yet been received). Thus, the receiver can keep track of tokens for which it has received the data packet (removing them from the window) and tokens for which it has not yet received the data packet (used for requesting packet retransmission). The start of the window points to the token with the smallest packet sequence number for which the token has been sent but the data packet has not yet been received. Until the window is full, the receiver sends out one token per MTU transmission time. If the admitted packets from the matched sender are delayed (*e.g.*, due to oversubscription or high-priority short flows), the window may fill up. The receiver then sends one new token for every data packet received from the matched sender.

dcPIM's token clocking mechanism, combined with its matching-based design, efficiently handles congestion due to oversubscription or high-priority short flows: since the number of tokens for which a data packet has not been received is bounded by the token window size, once the window fills up, the rate at which any receiver sends out tokens perfectly matches the rate at which data packets are received by the receiver. This has a desirable effect similar to many other window-based mechanisms: if data packets are delayed due to congestion in the core, the receiver will also delay sending out additional tokens; as the congestion alleviates, more data packets are received resulting in more tokens being sent out by the receiver.

Sender-side data transmission logic. dcPIM sender-side data transmission logic is also simple. Every MTU transmission time,

the sender checks if it has an outstanding short flow; if so, the sender transmits a packet from a short flow using the second highest priority. Otherwise, the sender checks if it has a token from the receiver it is currently matched with, if any, and transmits the data packet corresponding to that token. If the remaining flow size information for flows is available, it can be used by the sender to choose (among concurrently active flows) the next packet to transmit, and to set in-network priorities similar to [18, 34].

dcPIM senders use cumulative acknowledgements in tokens to clear up sender-side buffer; importantly, duplicate cumulative acknowledgements do not trigger data retransmission in dcPIM. The senders discard unused tokens at the end of the current data transmission phase, after a grace period of half of the control packet RTT (this grace period allows senders to efficiently use tokens which are sent by the receiver at the end of data transmission phase). Discarding unused tokens is important to avoid senders sending data packets for stale tokens; packets corresponding to discarded tokens will be sent when the sender is next matched with the receiver.

Fast retransmission mechanism upon data packet loss. dcPIM can experience data packet drops due to several reasons—contention between short flows (that are transmitted without explicit admission control), *e.g.*, in extreme incast traffic scenarios; contention between high-priority short flows and long flows; contention between long flows, *e.g.*, in oversubscribed topologies; and inevitable failures. dcPIM receivers detect data packet loss either using the `notification` packet (for short flows, this packet informs the receivers that packets are en route) or using the token window mechanism (that allows receivers to keep track of data packets for which tokens have been sent but the data has not been received). If packets from long flows are dropped, dcPIM simply retransmits the tokens at the start of the window upon subsequently matching with the sender. If packets from short flows are dropped, dcPIM requires the outstanding packets in the flow to be explicitly admitted (similar to [24])—the flow will participate in the next matching phase and packets need to be admitted by the receiver after matching. Intuitively, data packet drops in dcPIM indicate high congestion due to short flow incast and/or failures; thus, this is the right course of action. Since dcPIM prioritizes short(er) flows during the matching phase, the flows will not need to wait for too long before matching and subsequent retransmissions. dcPIM can be easily integrated with drop notification mechanisms [11, 45] for even faster short flow packet retransmissions, if datacenter network hardware supports such functionality.

3.3 Pipelining Phases

Datacenter networks have significantly larger RTTs when compared to switch fabrics. dcPIM hides these long RTTs by exploiting the fact that datacenter networks are rarely run at an average load of 100%, thus providing enough bandwidth to simultaneously send data and control packets. Specifically, dcPIM *pipelines* the current data transmission phase with the matching phase for the next data transmission phase (Figure 2). This way, data transmission continues without any disruption: the next data transmission phase can start right after the current data transmission phase ends.

Pipelining of matching and data transmission phases means that ideally the two phases would be completely aligned, that is, they

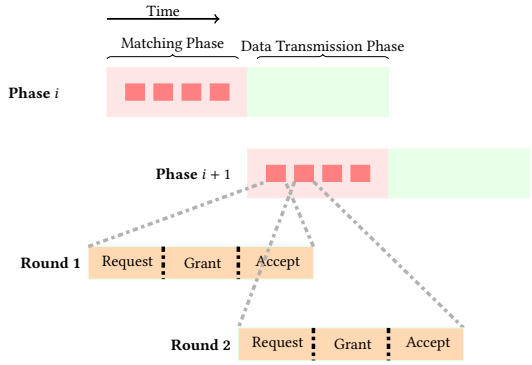


Figure 2: An illustration of dcPIM's pipelining (§3.3)

will have the same lengths. How should dcPIM choose the length of each phase? A longer length would allow more matching rounds, potentially increasing the matching size (thus, potentially higher utilization). However, if the length is too long, flows would not have enough packets to transmit during the data transmission phase, potentially resulting in lower utilization. dcPIM finds a sweet spot between the two extremes using two ideas:

Embracing PIM's simplicity: fixing the time for each stage. Recall from §2 that each round of matching uses three stages—request, grant and accept. The length of each stage determines the amount of time hosts wait to receive requests, grants and accept control packets, depending on the stage. dcPIM sending control packets at the highest priority and the network performing packet spraying means minimal queueing delay for control packets. Thus, in the common case, each stage would take half of the control packet RTT, or cRTT/2. However, to account for multiple control packets contending at the receiver downlink, dcPIM uses a small "slack" similar to PIM: it sets the length of each stage to be $\beta \times \frac{\text{cRTT}}{2}$, where β allows accounting for control packet queuing delays. Similar to PIM, it is okay for control packets to be delayed beyond this fixed time for two reasons: first, all control packets carry a round number, and straggler control packets can be ignored; and second, similar to PIM, randomized and multi-round nature of dcPIM naturally handle control packets being delayed beyond the fixed time since hosts unmatched in one round will be able to catch up in subsequent rounds. Thus, dcPIM recommends using a small value of β (in our evaluation, we set $\beta = 1.3$). Our sensitivity analysis in §4 suggests that any value larger than 1.1 has minimal impact on overall performance. Note that, for topologies where different pairs of hosts have different cRTTs, dcPIM decides the length of each stage based on the longest cRTT in the network.

Pipelining stages across multiple rounds. Each matching round requiring three stages (request, grant, accept) means that r rounds of matchings require executing $3 \cdot r$ stages; with each stage taking $\beta \times \frac{\text{cRTT}}{2}$ time, the total length of the matching and consequently the data transmission phases turns out to be $3r\beta \times \frac{\text{cRTT}}{2}$. We observe that it is possible to further reduce this length by pipelining the accept stage at the end of each round with the request stage at the beginning of the next round. Specifically, hosts that participate in accept and in request stages are completely different: in

the former, only matched receivers send `accept` packets and in the latter, only unmatched receivers send `request` packets. Thus, dcPIM aligns the accept stage of each round with the request stage of the next round, hence starting the next round earlier (Figure 2). When an unmatched sender receives an `accept` packet, it discards all other `request`s from the overlapping request stage of the next round and marks itself as matched. With such pipelining, it is easy to see that dcPIM needs to execute $2r + 1$ stages for the matching phase thus requiring the matching phase length to be $(2r+1)\beta\frac{\text{cRTT}}{2}$.

3.4 Sustaining Higher Loads

In dcPIM design described so far, once a receiver is matched to a sender, it admits packets only from that sender. That means matchings correspond to network utilization only if the sender has enough packets to be admitted by the receiver to fill up the entire data transmission phase. Consider, for instance, one of the topologies used in our evaluation: a leaf-spine datacenter network topology with 100Gbps links, unloaded RTT for data packets being 5.8μs, unloaded RTT for control packets being cRTT = 5.2μs, and bandwidth-delay product of 72.5KB. Consider dcPIM uses $r = 4$ rounds and slack value of $\beta = 1.3$; then, the length of the data transmission phase will be $(2r+1)\beta\frac{\text{cRTT}}{2} = 30.4\mu$s. Thus, if the sender has a flow of size 73KB (this is smallest size for a flow to require matching in dcPIM), it will transmit the entire flow within roughly the first RTT (5.8μs to be precise) resulting in both the sender and the receiver being idle for rest of the data transmission phase. As a result, in the worst-case scenario of flow sizes being just slightly larger than 1 BDP, both the sender and the receiver may be idle for $(1 - (5.8/30.42)) \approx 0.81$ fraction of the data transmission phase, potentially missing the opportunity to send or receive data from other idle hosts.

Instead, to make the most of the available bandwidth in the data transmission phase, dcPIM allows each receiver to be matched with more than one sender (and vice versa) in the matching phase, carefully regulating how multiple senders share the available bandwidth to the same receiver during data transmission. Conceptually, each receiver (or sender) divides its available bandwidth into k *channels* and matches with other senders (or receivers) upon a per-channel basis. Each channel operates using $1/k$ of the link bandwidth and $1/k$ of the token window (the problem of designing a near-optimal matching algorithm that performs non-uniform bandwidth allocation across channels is explored in [1]). Thus, if a receiver matches with k senders in each matching round, the flow size required to fill up the data transmission phase (for each channel) reduces by a factor of k; this does result in slightly higher latency for medium and long flows (since, as discussed below, they are transmitted at $1/k$ of the link bandwidth), but allows dcPIM to fill up the pipes more efficiently. In our example above, $k = 4$ allows achieving ~99% utilization even for the worst-case of flows of size 73KB. We provide details on this extension for dcPIM below.

The matching phase. The matching logic is largely similar to the case of $k = 1$ (§3.1) but with some extra book-keeping: the receiver keeps track of the number of outstanding bytes from each sender, and uses this information to compute the maximum number of channels needed to finish transmitting the outstanding bytes. In the grant stage, each sender picks a subset of the channels it has received requests for such that the sender's total number of matched channels do not exceed k. Similarly, in the accept stage, each receiver picks a subset of the channels it has received grants for such that the receiver's total number of matched channels do not exceed k. Finally, for each channel that is accepted from a sender, the receiver updates the number of outstanding bytes from that sender to account for the number of bytes that will be sent over the matched channels during the data transmission phase.

The data transmission phase. All k channels of a receiver (or a sender) share the link bandwidth in the data transmission phase, each using $1/k$ of the link bandwidth. To account for this, the receiver appropriately scales the time at which per-packet tokens are sent (based on the number of channels matched with each sender). If flow size information is available, priorities can also be set intelligently: the receiver can use tokens to communicate priorities for the packets from matched senders, with senders having flows with fewer remaining bytes being assigned higher priorities.

3.5 Asynchronous Design & Optimizations

We have already described most of the interesting aspects of dcPIM design in previous subsections. In this subsection, we outline a few more implementation details and optimizations incorporated within dcPIM to optimize latency and to handle control packet drops during inevitable failures.

Asynchronous design. Our design and implementation of dcPIM can benefit from but does *not* necessitate clock synchronization across hosts. Each host maintains its local view of the clock, and uses its local view of the stage/round/phase to make matching and data transmission decisions. Datacenter networks already use time synchronization protocols (*e.g.*, IEEE1588 Precision Time Protocol (PTP)) to achieve sub-microsecond network-wide synchronization; and most modern NICs support PTP. Moreover, as discussed earlier, the randomization and multi-round nature of PIM ensures that hosts unmatched in one round (*e.g.*, due to control packet delays) will be able to catch up in the remaining rounds, and will continue to request matching until the entire data transmission is complete.

Optimizing for latency. dcPIM analysis shows that it is possible to achieve near-optimal utilization if senders and receivers select requests and grants uniform randomly. dcPIM aims to also optimize the latency of smaller flows while keeping PIM's high utilization guarantees. As such, dcPIM allows extremely short flows ($\leq$ 1 BDP) to be sent without participating in matching. To further optimize latency for medium-sized flows, dcPIM adds a simple optimization to PIM's matching algorithm: it performs first round of matching based on flow sizes, if such information is available (if flow sizes are not known in advance, the matching in the first round boils down to random choice as in other rounds). Specifically, in the first round of the matching phase, senders pick the `request` with the smallest remaining flow size in the grant stage, and receivers pick the `grant` with the smallest remaining flow size in the accept stage; starting with the second round, senders and receivers pick `request` and `grant` packets uniform randomly, just like PIM. Using such a FCT-optimizing round allows dcPIM to achieve most of the benefits of protocols such as pHost [18] and Homa [34] that approximate the shortest remaining processing time first scheduling policy to optimize average flow completion times. The subsequent

utilization-optimizing rounds allow dcPIM to increase the matching size, ensuring high utilization.

Handling control packet loss. Recall that dcPIM sends all control packets at the highest priority. Thus, even with commodity switches, the network fabric behaves like a lossless fabric for control packets in failure-free scenarios. To handle control packet drops in presence of failures, dcPIM uses the observation that control packets being sent at the highest priority means that the end-to-end latency of a control packet in failure-free scenario is roughly equal to a control packet RTT in an unloaded network (since transmission and queueing delays for control packets are tiny). dcPIM exploits this observation as follows: if the sender does not receive an `ack` for the `notification` packet or a receiver-side `finish` for its own `finish` packet, the sender retransmits these control packets within an RTT and continues doing so until it receives the corresponding `ack` or `finish` packets or a data packet retransmission request. Dropped `token` packets are treated as dropped data packets and dcPIM recovers from their loss using the token clocking mechanism described in §3.2.

Handling control packet drops for the matching phase is even easier. As discussed earlier, if `grant` or `request` packets are lost in one round, their corresponding senders and receivers can catch up in the following rounds. If an `accept` packet is lost, its corresponding sender will consider itself as free while the receiver considers itself as matched; thus, the sender might match to other receivers in the following rounds, and receive tokens from multiple receivers in the data transmission phase. However, this rarely happens consistently in multiple consequent phases due to the inherent randomization in dcPIM's matching protocol.

3.6 dcPIM parameters: r, β and k

dcPIM design has three parameters: r, β and k. We already discussed, in §3.1, that r should be chosen based on Theorem 1 and desired network utilization/load. We also discussed in §3.3 that β should be very small since it is accounting for the rare queueing delay seen by control packets. The only remaining question is the tradeoff introduced by k.

Intuitively, a higher k will allow more senders and receivers to be matched during a matching phase, potentially increasing utilization. The case for flow completion time (FCT), however, is more nuanced: if more senders and receivers are matched, more flows can transmit data during the data transmission phase and can potentially finish faster. On the other hand, a higher k means that each channel has to share the available bandwidth with more flows and send fewer bytes during each data transmission phase, which can potentially delay the completion of some flows. The sweet spot is for k to be set equal to r, the number of matching rounds. To see why, consider the example from §3.4. When $k = 1$, the flows that hurt utilization the most are those that are slightly larger than 1 BDP. These flows have to be matched for transmission as they are larger than the short flow threshold, but will only transmit for $\sim$1 RTT in the transmission phase, which lasts for $\sim r$ RTTs. When k is equal to r, k flows can concurrently transmit for the entire transmission phase. Thus, $k = r$ is the sweet sport that provides significant utilization gains without considerably hurting FCT (§4).

4 EVALUATION

We evaluate dcPIM via packet-level simulations and via an end-to-end implementation running on a 32-node CloudLab testbed. We will show throughout the evaluation that:

dcPIM consistently achieves good short flow latency and network utilization. Across a variety of settings that mix-and-match three network topologies, three workloads, three traffic patterns, and varying network loads, dcPIM consistently achieves average and tail short flow latency close to hardware latency while sustaining significantly higher loads. We will also demonstrate, via microbenchmarks, that dcPIM's theoretical foundation allows it to achieve good utilization even for extreme workloads.

Reasons for dcPIM's performance. As we will show, state-of-the-art datacenter transport designs make one of the two design choices: they either give up on short flow latency by proactively dropping packets (*e.g.*, Homa Aeolus and NDP) or they give up on utilization by proactively ensuring that buffers do not fill up (*e.g.*, Homa and HPCC). dcPIM, using a matching-based design, breaks this hard tradeoff: in the absence of short flows, matching-based design and per-packet load balancing used in dcPIM ensures that queueing induced by long flows is negligible. Indeed, when short flows compete with long flows, dcPIM maintains low latency by prioritizing short flows, but does so with minimal impact on utilization—matching-based design ensures that buffers are filled only when matched flows compete with high-priority short flows, and token window based design ensures that maximum buffering is exactly 1BDP, precisely what is needed to keep the downlink busy for the next RTT. Thus, the main take-away is not that dcPIM achieves near-optimal average and tail latencies, but rather that dcPIM does so while achieving near-optimal network utilization.

4.1 dcPIM Simulation Results

We incorporate dcPIM within the pHost simulator [18]. The workloads, topologies, performance metrics, and protocols used in our simulations are summarized in Table 1. For all evaluated protocols, we use their respective simulators but ensure consistency in topology and workload settings.

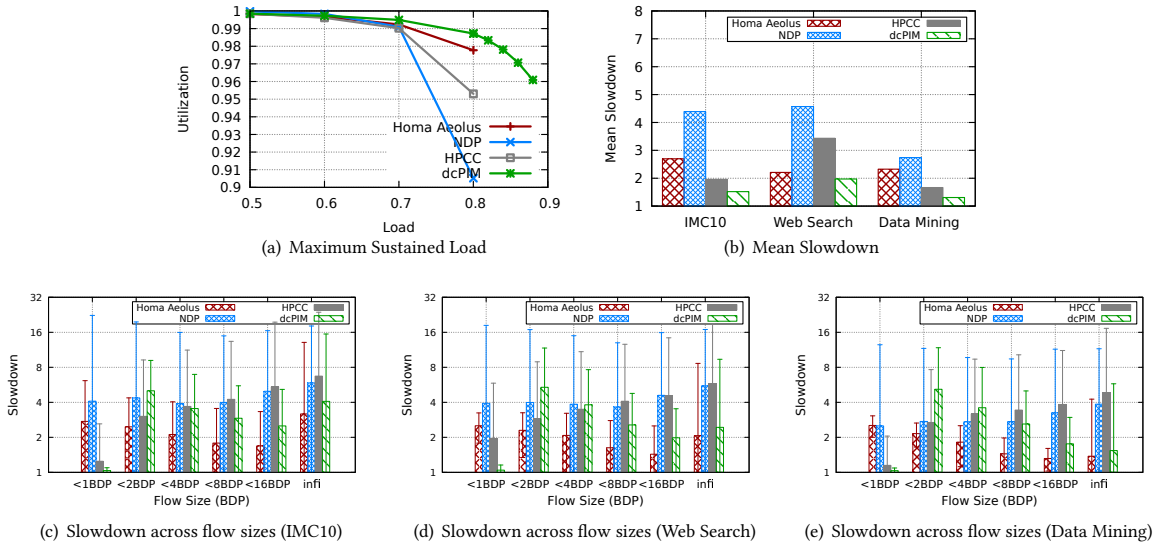
Default setup. Unless stated otherwise, we use the standard setup from prior work [7, 22, 24, 34]: all-to-all traffic pattern generated to create 0.6 load (maximum load sustainable by all protocols) over the leaf-spine topology. For dcPIM, we use one FCT-optimizing round, three utilization-optimizing rounds, $k = 4$ (number of channels), $\beta = 1.3$, and similar to prior work [18, 22, 24, 34], 1 BDP as short flow size threshold. For other protocols, we set the parameters suggested in the respective papers. See Table 1 for details.

dcPIM achieves near-optimal tail latency *and* network utilization. Figure 3 shows that existing protocols either achieve low latency for short flows or high utilization, but not both. dcPIM, on the other hand, is able to simultaneously achieve both. We provide intuitive reasons for each protocol individually.

Homa Aeolus achieves the best network utilization among existing protocols, coming closest to dcPIM (Figure 3(a)), and good average slowdowns across all flows (Figure 3(b)). However, it does

Topologies	- Two-tier leaf-spine, commonly used in the literature [7, 18, 34]: 4 spines, 9 racks each with 16 end-hosts - FatTree: Three-tier 1024 end-hosts - Oversubscribed: same as two-tier leaf-spine but with 2:1 oversubscription ratio.
Workloads	- IMC10 [18], Web Search [7], and Data Mining [7]
Traffic Patterns	- All-to-All: each sender to a random receiver (generated using a Poisson process from the workloads) - Bursty: all-to-all traffic pattern + a 50:1 incast workload [28] - Dense traffic matrix : 144×143 flows with each sender having a flow for each receiver
Evaluated Protocols	- Homa Aeolus [24, 34] and NDP [22]: Receiver-driven transport protocols - HPCC [5]: Rate-control using precise link load information from the network (through INT)
Evaluated Metrics	- Slowdown: ratio of observed FCT to the optimum FCT (when the flow is the only one in the network) - Utilization: ratio of the protocol's achieved throughput and the offered load
Link Properties	- 100Gbps access links for end-hosts, 200ns propagation delay - 400Gbps links between switches in leaf-spine, and 100Gbps in FatTree
Switch Properties	- 500KB per port buffer or 16 MB switch buffer (as common in prior work [7, 18]) (for NDP, we use small per-port buffer (8 packets) to enable timely packet trimming for fast retransmission.) - 450ns processing latency and packet spraying

Table 1: Summary of evaluation scenarios and parameters for simulations.



(a) Maximum Sustained Load

(b) Mean Slowdown

(c) Slowdown across flow sizes (IMC10)

(d) Slowdown across flow sizes (Web Search)

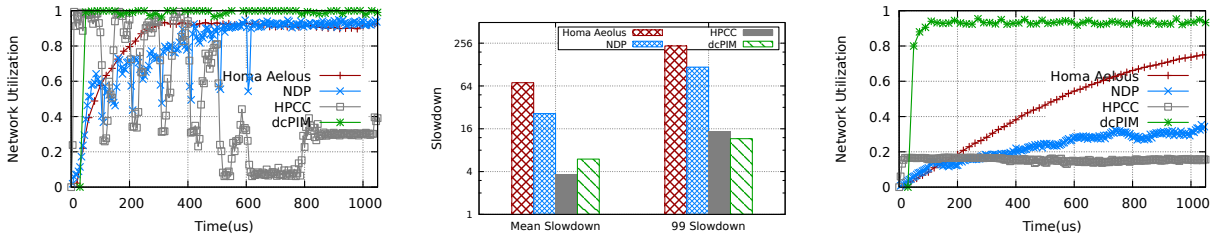
(e) Slowdown across flow sizes (Data Mining)

Figure 3: Evaluation results for the default setup. Figure 3(a) demonstrates the maximum load sustained by each protocol for the IMC10 workload. Figure 3(b) shows the mean slowdown across all flows for load 0.6 (maximum load that all protocols can sustain). Figures 3(c) to 3(e) show the mean and 99th-percentile slowdown broken down by flow size (x-axis labels).

so by trading off short flow latency (Figure 3(c), 3(d), 3(e)): we observe short flow slowdown of 2.5–2.7 on an average, and 3–6.1 at tail across various workloads. The reason is that Homa Aeolus prioritizes packets scheduled by the receiver; thus, unscheduled short flows may be dropped even if there is one link along the path that is transmitting scheduled packets; thus, average and tail latencies suffer. Recall that Homa Aeolus builds upon Homa; the original Homa evaluation was performed with infinitely large switch buffers, and uses 10Gbps links. Our own evaluation confirms the observations made in Homa Aeolus: when realistic buffer sizes are used, Homa suffers from significant packet drops and low network utilization, especially for 100Gbps links. Thus, for sustainable loads, Homa will achieve near-optimal short flow latency (similar to dcPIM, since we use exactly the same prioritization mechanism for short flows) but will suffer from degraded network utilization. Homa Aeolus makes a better tradeoff when compared to Homa: give up a bit on short flow latency to achieve much better utilization.

NDP suffers from high short flow latency as well as poor network utilization. We observe short flow slowdown of 2.5–4.1 on an average, and 12.5–22.3 at tail across various workloads. The reason is two-fold: (1) similar to Homa Aeolus, NDP proactively drops packets to maintain low queue occupancy; and (2) NDP does not use prioritization. NDP's proactive dropping of packets combined with a lack of mechanism to ensure that retransmitted packets are not dropped again also results in poor utilization.

HPCC is an interesting case. HPCC performs aggressive rate control to ensure minimal queue occupancy at each switch. This allows HPCC to achieve very low latency for short flows—we observe slowdown of 1.1–1.9 on an average and tail slowdown of 2–5.8 across workloads. This is because senders transmit short flows at



(a) dcPIM robustness in bursty workloads (b) dcPIM worst case: all flows of size BDP+1 (c) dcPIM bad case: dense traffic matrix ($\bar{\delta} = 144$)

Figure 4: Microscopic view into dcPIM performance. (left) even under bursty workloads (all-to-all traffic combined with periodic incast), dcPIM maintains high network utilization. (center) for the hypothetical case of all flows of size BDP+1 (worst-case for dcPIM, §3.4), dcPIM has slightly higher latency than HPCC; and (right) dcPIM achieves high network utilization even when the traffic matrix is not sparse. Discussion in §4.1.

full rate, and little to no queueing at each switch ensures near-zero queueing delays. However, such aggressive focus on keeping low queues also means that HPCC suffers from suboptimal utilization: temporary queueing caused by short flows results in long flows reducing their rate and taking one extra RTT to ramp back up.

dcPIM is able to achieve near-optimal average and tail latency for short flows, while maintaining high network utilization. For example, across workloads, we observe average short flow slowdown of 1.03–1.04 (as much as 2.6×, 4× and 1.8× better than Homa Aeolus, NDP and HPCC, respectively) and tail short flow slowdown of 1.09–1.16 (as much as 5.6×, 20.5× and 5× higher than Homa Aeolus, NDP and HPCC, respectively). The fact that dcPIM achieves lowest average and tail latency for short flows is not surprising: it always prioritizes short flows over long flows; what may be surprising is that it achieves such low latency without impacting network utilization. For instance, Figure 3(a) shows that dcPIM can sustain network loads as high as 0.84 for the IMC10 and Web Search workloads, and as high as 0.7 for the Data Mining workload.

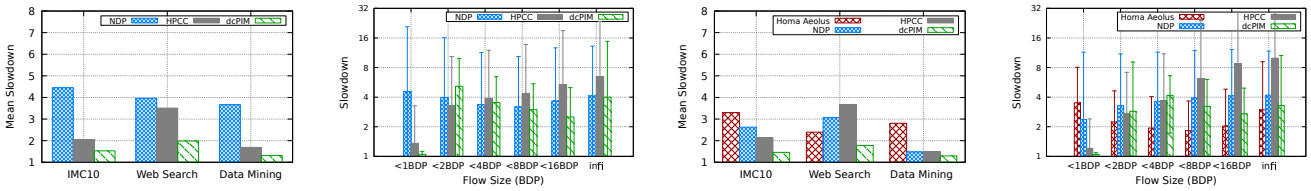
How is dcPIM able to maintain high network utilization? We already summarized the reasons in the beginning of the evaluation section (the second takeaway), but provide a little more detail here. There are three reasons for this. First and foremost, just like PIM achieves much better switch fabric utilization than its theoretical bound, dcPIM achieves much better network utilization than its theoretical bound. Second, dcPIM mostly performs matching for flows that are at least longer than the short flow threshold; since extremely short flows are not accounted for in utilization, the flows transmitted after matching have enough packets to utilize the entirety of the data transmission phase (after all, most bytes in datacenter traffic are contained in long flows [9, 19, 37]). Finally, the core reason is very conceptual: as discussed above, dcPIM buffers up packets in the queues only when long flow compete with high-priority short flows; and the amount of buffering is just "right": the token clocking mechanism ensures that, at any point of time, one BDP worth of long flow packets are in flight, exactly the amount needed to keep the link busy for the next round trip time.

dcPIM maintains low latency for short flows without impacting network utilization by making a specific tradeoff: latency of medium-sized flows; this is because the time taken to match flows before they can be transmitted incurs the most overhead for flows that are not too short and are not too long. We believe this is the right tradeoff to make for the following reason. It is not too hard to see that achieving high network utilization requires maintaining low average latency for long flows: for these flows, throughput is given by the ratio of flow size and flow completion time. Thus, if the goal is to achieve low tail latency for short flows and low average latency for long flows, any datacenter transport design must tradeoff the flow completion time for medium size flows (other near-optimal datacenter protocol designs also make a similar tradeoff [2]). If necessary, network operators can explore this tradeoff space by choosing any desired threshold for short flows and any desired value of number of matching rounds and number of channels (we perform sensitivity analysis later in the evaluation).

Microscopic view into dcPIM benefits. Large-scale simulations often hide how protocols handle short-term traffic bursts. In our next experiment, we setup a microbenchmark: on the same two-tier leaf-spine topology, we have 16 senders in the same ToR send an all-to-all traffic to 16 receivers in the other ToR (*e.g.*, a MapReduce application executing a shuffle), and every 100μs for the first 600μs, 50 other senders send 128KB flows as an incast traffic to one of the above receivers (*e.g.*, a parameter-server based application colocated with the MapReduce application). Figure 4(a) shows that HPCC stumbles under such a workload due to frequent triggering of PFC. Homa Aeolus and NDP converge to relatively good utilization but take $300 - 600\mu$s to converge due to poor matching—multiple receivers repeatedly send grants to the same sender, which can respond to only one receiver's grant at a time, thus resulting in some receivers being idle and subsequent underutilization. dcPIM's matching algorithm operates at tens of μs; thus, it not only converges quickly but also achieves high network utilization by computing good matchings[3].

Worst-case for dcPIM: understanding dcPIM limitations. We have tried hard to find a scenario where dcPIM performs worse than state-of-the-art datacenter protocols. We have found one scenario: all-to-all traffic with 0.6 load on a 144-node leaf-spine topology as in the default setup but with one change: rather than using the Web Search workload, we use a workload that has all flows of size BDP+1. While unrealistic, this workload helps us demonstrate the limitations of dcPIM. Intuitively, for this workload, dcPIM will send each flow only after the sender is matched with the corresponding receiver, thus incurring latency overhead for each flow. Figure 4(b)

[3]Since the first matching phase cannot be pipelined by the data transmission phase, the network utilization of dcPIM is 0 during the first matching phase.



(a) Oversubscribed Slowdown (b) Oversubscribed Slowdown per Flow Size (c) FatTree Slowdown (d) FatTree Slowdown per Flow Size

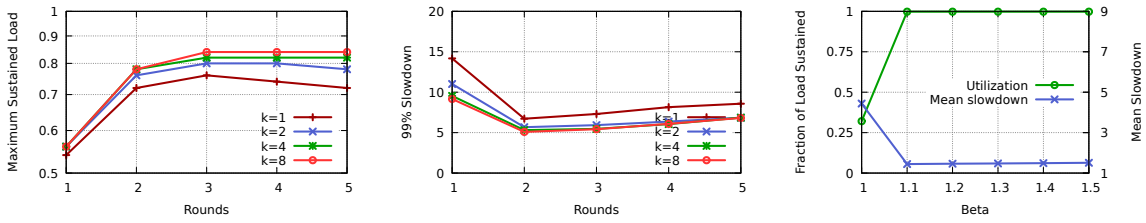
Figure 5: (left and center left) results for oversubscribed topology demonstrating the effectiveness of dcPIM's token clocking mechanism in gracefully responding to potential congestion in the core. (right and center right) results for a 1024-node FatTree topology. * we were unable compare against Homa Aeolus in oversubscribed topologies. See §4.1 for detailed discussion.

shows the results: we observe that, for this synthetic workload, HPCC is able to achieve better average latency and slightly better tail latency than dcPIM; NDP and Homa Aeolus continue to achieve worse average and tail latency due to proactively dropping packets when more than one flow compete on any outgoing link. If a network operator expects to run the network over such a workload, they can easily tune the short flow threshold used in dcPIM to account for this worst-case scenario.

What if the sparse traffic matrix assumption does not hold? As discussed earlier, datacenter traffic analysis studies have shown that the *average* incast ratios in production datacenters are small. However, it may happen that incast ratios may be large over a subset of the datacenter over a short period of time (*e.g.*, in a large-scale MapReduce deployment executing shuffle). To evaluate dcPIM performance over such workloads, we create another microbenchmark where each of the 144 senders have a long flow to send to each of the 144 receivers (total number of flows = 144×144), and we measure the overall network utilization. Figure 4(c) shows that existing protocols actually achieve low utilization for this workload. HPCC suffers from PFC triggering all the time due to congestion being both in the core and at the receiver (as HPCC paper notes, it may suffer from poor performance when there are multiple congestion points); NDP suffers from large number of retransmissions due to aggressive dropping and from not handling in-network congestion; and, Homa Aeolus suffers from convergence time—it converges to the right matching due to prioritizing scheduled flows, but takes more than 1000μs to converge. dcPIM performance degrades compared to previous results, but it still achieves ~93.5% utilization. This is surprising; we found the maximal matching is often of very large size for this workload ($M^\star \approx 120$); thus, with $N = 144, \bar{\delta} = 144, \alpha = (144/120) = 1.2, r = k = 4$, Theorem 1 gives us an expected matching size and network utilization of 32.9% in any matching phase. Digging deeper into this result, we found that dcPIM high performance is due to dcPIM converging to matching sizes that are larger than what the theoretical bound suggests (as is seen during the first ~100μs in the figure)—this is not surprising since our theoretical bound is on expected matching size, where the randomization is over sequence of `grant` and `accept` decisions made by senders and receivers, respectively, and it is hard to generate the precise bad-case sequences.

Additional topologies and workloads. Figure 5 presents results for oversubscribed and FatTree topologies. For the former, we use the same leaf-spine topology as earlier, but decrease the bandwidth between leaf and core switches from 400 Gbps to 200Gbps to create an oversubscription ratio of 2:1. We generate all-to-all traffic traces from all three workloads for a network load of 0.5; we chose load 0.5 because, unfortunately, none of the NDP, HPCC and Homa Aeolus can sustain any higher loads—NDP continues to drop retransmitted packets, do fast retransmission, and again drop retransmitted packets; HPCC results in high PFC trigger rates, and Homa Aeolus has no mechanism to handle drops of scheduled packets. For both oversubscribed and FatTree topologies, the observed results have the same trend as in Figure 3. This is because, in the oversubscribed topology, dcPIM's token clocking mechanism enables receivers to carefully admit packets in presence of in-network congestion; and, in the FatTree Topology, although dcPIM sets the length of matching phase based on longest RTT as we discuss in §3.5, pipelining of matching and data transmission phases hides large datacenter round trip times. dcPIM evaluation over several additional workloads, including a mix of all-to-all traffic with bursty incast traffic [28], consistently exhibits similar performance.

Sensitivity Analysis of dcPIM Parameters. Finally, we evaluate the sensitivity of dcPIM performance to its three parameters—number of matching rounds (r) and channels (k), and slack (β)—using our default setup with one change: we use a network load of 0.54 rather than 0.6 as that is the highest load that dcPIM can sustain with all combinations of evaluated parameters. Note that for $r = 1$, we only have one FCT-optimizing round and no utilization-optimizing rounds, and therefore our theoretical analysis does not hold. Figure 6 shows the results. We observe that going from 1 to 2 rounds has the most significant impact since dcPIM's matching algorithm kicks in—we observe 18 – 24% higher sustainable load with two rounds (additional rounds result in diminishing returns); improved utilization also results in improved tail latency. As expected, increasing the number of rounds results in slightly higher sustainable loads, but at the cost of latency increase (since flows wait longer before they can start data transmission). Using 2–4 channels gives dcPIM the best tradeoff: we observe higher utilization because more channels allow dcPIM to do much more fine-grained matching making it less likely for matched senders and receivers to become idle during data transmission (§3.4). We observe that β has no impact beyond 1.1: as discussed in §3.3, β is a safety guard for rare queueing of control packets; thus, it impacts neither latency nor utilization for dcPIM.



(a) Maximum Sustained Load (r, k) (b) 99% Slowdown (r, k) (c) Sustained Load & Mean Slowdown (β)

Figure 6: dcPIM sensitivity analysis against number of rounds (r) and channels (k), and slack (β). See discussion in §4.1.

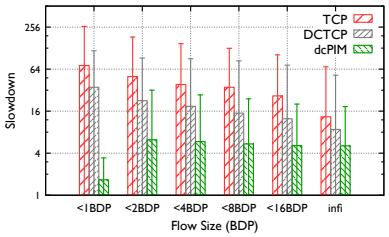


Figure 7: 32-server CloudLab testbed results for dcPIM, DCTCP and TCP cubic: mean and tail slowdown (error bars show 99%) across different flow sizes. y-axis is log-scale. See §4.2.

4.2 dcPIM Implementation Results

We have implemented a dcPIM prototype in Linux using DPDK [15], using ~3K lines of C code. We use CloudLab to evaluate the current dcPIM implementation over a 32-server two-tier leaf-spine topology (10Gbps links and ~8μs RTT).

Before discussing implementation results, we make two important notes. First, the current dcPIM implementation is not optimized for CPU efficiency and requires more work to alleviate network stack overheads [10]; thus, we focus on measuring latency and network utilization rather than CPU efficiency. Second, as discussed in §3.5, dcPIM can operate upon loosely synchronized host clocks (*e.g.*, using time synchronization protocols like PTP [13]); however, our CloudLab testbed does not support PTP. We expect dcPIM to perform even better in datacenter networks that typically employ time synchronization protocols.

Figure 7 shows the average and tail slowdown for dcPIM, DCTCP and TCP Cubic over CloudLab testbed for the same all-to-all traffic pattern as earlier, but using load 0.5. The results are as expected: for short flows, dcPIM achieves 21 – 43× better average slowdown and 34 – 76× better $P99$ tail slowdown than DCTCP and TCP, while sustaining 1.71 – 2.61× higher throughput for long flows (1.71 – 2.61× lower flow completion time for long flows).

5 RELATED WORK

There is a large and active body of research on datacenter transport protocols [4–6, 12, 21, 23, 35, 39, 46, 47]; it would be a futile attempt to compare dcPIM with each and every protocol. In our evaluation, we have already evaluated state-of-the-art rate control and scheduling based transport designs, and demonstrated that dcPIM design avoids the fundamental tradeoff between short flow latency and network utilization in prior designs. One design that we did not present results for is Fastpass [36]. Fastpass uses a centralized scheduler to not only match senders and receivers, but also to compute the network paths between them for data transmission. Using a centralized scheduler allows Fastpass to get good utilization; however, Fastpass not only suffers from scalability issues for modern high-bandwidth networks, but also from high short flow latency: since all short flows need to be scheduled before transmission, their average and higher tail latency is at least 2× away from optimal; dcPIM achieves much better short flow tail latency.

The use of matching mechanisms has a long history in switch scheduling [8, 25, 26, 31–33, 40, 41, 44], to name a few. dcPIM places its intellectual roots in PIM due to two reasons. First, as shown in a recent theory result, PIM provides the optimal tradeoff between number of rounds and matching size guarantees [29]; other protocols (*e.g.*, IRRM [33] and iSLIP [31]) provide better tradeoff by making assumptions on input workloads, but can perform much worse when these assumptions do not hold. PIM, on the other hand, makes no such assumption and has been shown to work well independent of underlying workload.

6 CONCLUSION

Modern datacenter networks bear a striking similarity to switch fabrics; yet, scheduling mechanisms in existing datacenter transport designs are significantly different from those used in switch fabrics. Bridging this gap has the potential for datacenter transport designs to benefit from decades of foundational work on switch scheduling that has led to near-optimal switch fabric designs. To that end, we have presented Datacenter Parallel Iterative Matching (dcPIM), a proactive receiver-driven transport design that places its intellectual roots in classical switch scheduling protocols to simultaneously achieve near-optimal tail latency for short flows and near-optimal network utilization, without requiring any specialized network hardware. We have demonstrated, both theoretically and empirically, the near-optimality of dcPIM performance.

ACKNOWLEDGMENTS

We would like to thank our shepherd, Zaoxing Liu, and anonymous SIGCOMM reviewers for insightful feedback. We would like to also thank Saksham Agarwal, Shijin Rajakrishnan and David Shmoys for many helpful discussions during the course of this project. This research was in part supported by NSF grants CNS-2047283 and CNS-1704742, a Google faculty research award, and a Sloan fellowship. This work does not raise any ethical issues.

REFERENCES

[1] Rachit Agarwal, Shijin Rajakrishnan, and David Shmoys. 2022. From Switch Scheduling to Datacenter Scheduling: Matching-Coordinated Greed Is Good. In *ACM PODC*.

[2] Saksham Agarwal, Shijin Rajakrishnan, Akshay Narayan, Rachit Agarwal, David Shmoys, and Amin Vahdat. 2018. Sincronia: near-optimal network design for coflows. In *ACM SIGCOMM*.

[3] Mohammad Al-Fares, Alexander Loukissas, and Amin Vahdat. 2008. A scalable, commodity data center network architecture. In *ACM SIGCOMM*.

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

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

[6] Mohammad Alizadeh, Abdul Kabbani, Tom Edsall, Balaji Prabhakar, Amin Vahdat, and Masato Yasuda. 2012. Less is More: Trading a Little Bandwidth for Ultra-low Latency in the Data Center. In *USENIX NSDI*.

[7] Mohammad Alizadeh, Shuang Yang, Milad Sharif, Sachin Katti, Nick McKeown, Balaji Prabhakar, and Scott Shenker. 2013. pFabric: Minimal Near-optimal Datacenter Transport. In *ACM SIGCOMM*.

[8] Thomas E Anderson, Susan S Owicki, James B Saxe, and Charles P Thacker. 1993. High-speed switch scheduling for local-area networks. In *ACM TOCS*.

[9] Theophilus Benson, Aditya Akella, and David A Maltz. 2010. Network traffic characteristics of data centers in the wild. In *ACM IMC*.

[10] Qizhe Cai, Shubham Chaudhary, Midhul Vuppalapati, Jaehyun Hwang, and Rachit Agarwal. 2021. Understanding Host Network Stack Overheads. In *ACM SIGCOMM*.

[11] Peng Cheng, Fengyuan Ren, Ran Shu, and Chuang Lin. 2014. Catch the Whole Lot in an Action: Rapid Precise Packet Loss Notification in Data Center. In *USENIX NSDI*.

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

[13] Richard Cochran, Cristian Marinescu, and Christian Riesch. 2011. Synchronizing the Linux system time to a PTP hardware clock. In *IEEE ISPCS*.

[14] Abhishek Dixit, Pawan Prakash, Yu Charlie Hu, and Ramana Rao Kompella. 2013. On The Impact Of Packet Spraying In Data Center Networks. In *IEEE INFOCOM*.

[15] DPDK. 2022. Data Plane Development Kit. http://dpdk.org/. (2022).

[16] Nathan Farrington, Erik Rubow, and Amin Vahdat. 2009. Data center switch architecture in the age of merchant silicon. In *IEEE Symposium on High Performance Interconnects*.

[17] Daniel Firestone, Andrew Putnam, Sambhrama Mundkur, Derek Chiou, Alireza Dabagh, Mike Andrewartha, Hari Angepat, Vivek Bhanu, Adrian Caulfield, Eric Chung, et al. 2018. Azure accelerated networking: Smartnics in the public cloud. In *USENIX NSDI*.

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

[19] Monia Ghobadi, Ratul Mahajan, Amar Phanishayee, Nikhil Devanur, Janardhan Kulkarni, Gireeja Ranade, Pierre-Alexandre Blanche, Houman Rastegarfar, Madeleine Glick, and Daniel Kilper. 2016. Projector: Agile reconfigurable data center interconnect. In *ACM SIGCOMM*.

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

[21] Matthew P Grosvenor, Malte Schwarzkopf, Ionel Gog, Robert NM Watson, Andrew W Moore, Steven Hand, and Jon Crowcroft. 2015. Queues don't matter when you can jump them!. In *USENIX NSDI*.

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

[23] Chi-Yao Hong, Matthew Caesar, and P Godfrey. 2012. Finishing flows quickly with preemptive scheduling. In *ACM SIGCOMM*.

[24] Shuihai Hu, Wei Bai, Gaoxiong Zeng, Zilong Wang, Baochen Qiao, Kai Chen, Kun Tan, and Yi Wang. 2020. Aeolus: A Building Block for Proactive Transport in Datacenters. In *ACM SIGCOMM*.

[25] Amos Israeli and Alon Itai. 1986. A fast and simple randomized parallel algorithm for maximal matching. In *Information Processing Letters*.

[26] Tara Javidi, Robert Magill, and Terry Hrabik. 2001. A high-throughput scheduling algorithm for a buffered crossbar switch fabric. In *IEEE ICC*.

[27] Richard M Karp, Umesh V Vazirani, and Vijay V Vazirani. 1990. An optimal algorithm for on-line bipartite matching. In *ACM STOC*.

[28] Yuliang Li, Rui Miao, Hongqiang Harry Liu, Yan Zhuang, Fei Feng, Lingbo Tang, Zheng Cao, Ming Zhang, Frank Kelly, Mohammad Alizadeh, and Minlan Yu. 2019. HPCC: High Precision Congestion Control. In *ACM SIGCOMM*.

[29] Zvi Lotker, Boaz Patt-Shamir, and Seth Pettie. 2015. Improved distributed approximate matching. In *ACM JACM*.

[30] M Ajmone Marsan, Andrea Bianco, Paolo Giaccone, Emilio Leonardi, and Fabio Neri. 2003. Multicast traffic in input-queued switches: optimal scheduling and maximum throughput. In *IEEE/ACM ToN*.

[31] Nick McKeown. 1999. The iSLIP Scheduling Algorithm for Input-queued Switches. In *IEEE/ACM ToN*.

[32] Nick McKeown, Adisak Mekkittikul, Venkat Anantharam, and Jean Walrand. 1999. Achieving 100% throughput in an input-queued switch. In *IEEE Transactions on Communications*.

[33] Nick McKeown, Pravin Varaiya, and Jean Walrand. 1993. Scheduling cells in an input-queued switch. In *Electronics Letters*.

[34] Behnam Montazeri, Yilong Li, Mohammad Alizadeh, and John Ousterhout. 2018. Homa: A Receiver-driven Low-latency Transport Protocol Using Network Priorities. In *ACM SIGCOMM*.

[35] Ali Munir, Ghufran Baig, Syed M Irteza, Ihsan A Qazi, Alex X Liu, and Fahad R Dogar. 2014. Friends, not foes: synthesizing existing transport strategies for data center networks. In *ACM SIGCOMM*.

[36] Jonathan Perry, Amy Ousterhout, Hari Balakrishnan, Devavrat Shah, and Hans Fugal. 2015. Fastpass: A centralized zero-queue datacenter network. In *ACM SIGCOMM*.

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

[38] Ahmed Saeed, Nandita Dukkipati, Vytautas Valancius, Carlo Contavalli, Amin Vahdat, et al. 2017. Carousel: Scalable Traffic Shaping at End Hosts. In *ACM SIGCOMM*.

[39] Ahmed Saeed, Varun Gupta, Prateesh Goyal, Milad Sharif, Rong Pan, Mostafa Ammar, Ellen Zegura, Keon Jang, Mohammad Alizadeh, Abdul Kabbani, and Amin Vahdat. 2020. Annulus: A Dual Congestion Control Loop for Datacenter and WAN Traffic Aggregates. In *ACM SIGCOMM*.

[40] Devavrat Shah, Paolo Giaccone, and Balaji Prabhakar. 2002. Efficient randomized algorithms for input-queued switch scheduling. In *IEEE Micro*.

[41] Devavrat Shah and Damon Wischik. 2006. Optimal scheduling algorithms for input-queued switches. In *IEEE INFOCOM*.

[42] Arjun Singh, Joon Ong, Amit Agarwal, Glen Anderson, Ashby Armistead, Roy Bannon, Seb Boving, Gaurav Desai, Bob Felderman, Paulie Germano, et al. 2015. Jupiter rising: A decade of clos topologies and centralized control in google's datacenter network. In *ACM SIGCOMM*.

[43] Robert Endre Tarjan. 1983. *Data structures and network algorithms*. Vol. 44. Siam.

[44] George Varghese. 2005. *Network Algorithmics: an interdisciplinary approach to designing fast networked devices*. Morgan Kaufmann.

[45] David Zats, Anand Padmanabha Iyer, Ganesh Ananthanarayanan, Rachit Agarwal, Randy Katz, Ion Stoica, and Amin Vahdat. 2015. FastLane: Making Short Flows Shorter with Agile Drop Notification. In *ACM SoCC*.

[46] Yibo Zhu, Haggai Eran, Daniel Firestone, Chuanxiong Guo, Marina Lipshteyn, Yehonatan Liron, Jitendra Padhye, Shachar Raindel, Mohamad Haj Yahia, and Ming Zhang. 2015. Congestion control for large-scale RDMA deployments. In *ACM SIGCOMM*.

[47] Noa Zilberman, Gabi Bracha, and Golan Schzukin. 2019. Stardust: Divide and Conquer in the Data Center Network. In *USENIX NSDI*.