Manuscript Under Submission

Your Datacenter is a Switch

Qizhe Cai
Cornell University

Mina Tahmasbi Arashloo
Cornell University

Rachit Agarwal
Cornell University

Abstract

Datacenter Parallel Iterative Matching (dcPIM) is a transport design that realizes the classical Parallel Iterative Matching (PIM) protocol from switch scheduling literature on datacenter networks without using any specialized hardware. Our key technical result is an extension of PIM's theoretical analysis to the datacenter context. dcPIM design builds upon insights gained from this analysis, by extending PIM to overcome the unique challenges introduced by datacenter environments (much larger scales and round trip times). Using theoretical analysis and evaluation, we show that dcPIM maintains near-optimal latency of state-of-the-art data center transport designs, sustains $6-10\%$ higher loads on evaluated workloads, and guarantees near-optimal network utilization.

1 Introduction

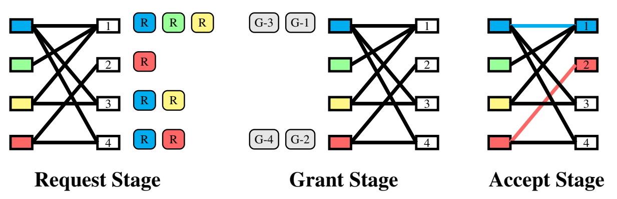
Modern datacenter networks bear a striking similarity to switching fabrics [5, 15, 19, 44]. Both of these are organized around Clos-like topologies using low-port count switches and both of these experience similar workloads — incast (many input ports having packets for the same output port), one-to-one (each input port having packets for one output port), all-to-all, etc. Yet, scheduling mechanisms in state-of-the-art datacenter transport designs [9, 12, 21, 23, 35, 38] are significantly different from those used in switching fabrics. Bridging this gap has the potential to allow datacenter transport designs to benefit from decades of foundational work on switch scheduling, that had led to optimal switch scheduling mechanisms for switching fabrics [10, 29, 30, 31, 43, 45].

Motivated by the above, we set out to explore an efficient datacenter-scale realization of Parallel Iterative Matching (PIM) [10], a time-tested protocol variations of which are used in almost all switching fabrics. The result of this exploration is Datacenter Parallel Iterative Matching (dcPIM), a transport design that efficiently realizes PIM on datacenter networks with commodity switches, offering them the many benefits of the PIM protocol that switching fabrics enjoy (including throughput-optimality), see §2.

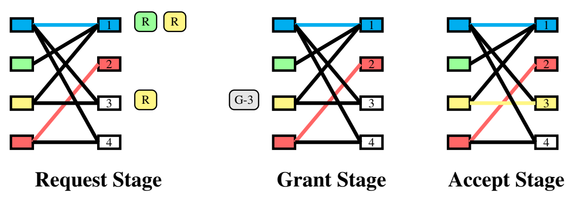
dcPIM places its intellectual roots in PIM [10]. Just like PIM, end-hosts in dcPIM exchange multiple rounds of control plane messages to "match" senders with receivers to ensure that at any point of time, each receiver admits packet from exactly one source. Just like PIM, dcPIM embraces simplicity: the number of rounds of control plane messages, the timescale for each control plane round, and the number of data packets that can be sent upon matching is fixed in advance. Finally, just like PIM, dcPIM embraces imperfection: it is okay for some of the control messages to be delayed and for some of the data packets to not complete transmission within the fixed specified time; the randomized matching algorithm in PIM combined with multiple rounds of control plane messages ensures that end-hosts unmatched in one round will be able to catch up in the remaining rounds (§3.1), and will continue to request matching until the entire data transmission is complete (§3.2).

What differentiates dcPIM from PIM is the environment: while PIM was designed for switching fabrics that have tens of ports and pico-second RTTs between unloaded ports, dcPIM needs to operate in a much harsher environment: datacenter networks have much larger scales and much larger RTTs between end-hosts. dcPIM resolves these challenges using two properties of datacenter environments. First, unlike switching fabrics where all input ports may very well want to send packets to a single output port, it is rarely the case that all end-hosts in the datacenter will have packets to send to a single end-host in the entire datacenter. That is, the underlying traffic matrix is sparse: several studies from production datacenter networks [11, 16, 18, 39, 40] show that the number of flows at any point of time, *when averaged across all end-hosts*, is a small constant (varying from $5-56$ across various studies). Second, unlike switching fabrics that are designed to run at "full" load, datacenter networks are rarely run at an average load of 100%.

dcPIM leverages the traffic matrix sparsity in datacenter networks to demonstrate theoretically that, unlike PIM that provides near-optimal utilization guarantees using $\log(n)$ rounds of control plane messages for a n-port switch fabric, it is pos-



(a) Round One



(b) Round Two

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

sible to achieve near-optimal utilization guarantees with constant number of rounds (independent of number of end-hosts in the network)! This allows dcPIM to scale to arbitrarily large datacenter network fabrics. Moreover, dcPIM leverages the fact that datacenter networks are rarely run at 100% load to carefully *pipeline* the control plane messages for next matching "phase" with data transmission of the current matching phase to hide the overheads of longer RTTs of datacenter networks. The result is a new 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 latency and network utilization.

We have implemented dcPIM in Linux hosts, and in simulations. Implementation results over a CloudLab testbed and simulation results show that dcPIM maintains its performance benefits over state-of-the-art datacenter transport designs [21, 23, 27, 35] across a variety of evaluation settings that mix-and-match three workloads, three traffic patterns, three network topologies, varying network oversubscription, varying access link bandwidths and varying network loads.

This work does not raise any ethical issues.

2 Parallel Iterative Matching (PIM) Overview

Most modern switches have multi-stage Clos interconnects: they can deliver data from multiple input ports to multiple output ports at line rate and without internal buffering as long as each output port receives data from at most one input port. Thus, scheduling the interconnect 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 [24, 26, 28, 45]. Among the many existing matching algorithms, PIM [10] has been widely adapted in switching fabrics for its simplicity and efficiency (we provide a comparison with other matching mechanisms in §5). PIM uses multiple rounds of control message exchange to match input and output ports. Specifically, 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 from top to bottom, 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 (notice the incast nature of the traffic). 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. These two output ports will mark themselves as matched while the rest stay unmatched, creating a matching of size 2.

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 pairs will be added to it in the subsequent rounds. Once the matching rounds are over, the matched input ports send data to their matched output ports.

PIM properties. PIM has several attractive properties that has led to the widespread adaptations of its variants in today's switching fabrics. First, it computes a maximal matching in $O(\log n)$ rounds, where n is the number of ports. In switching fabrics with only tens of ports (small n), 6-8 rounds are enough to converge to a maximal 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 `requests` and `grants` as in Figure 1), the entire computation of matching takes 12 RTTs. In switching fabrics, the round trip time 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, switching fabrics can simply execute matching and packet 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 — unmatched ports will get to catch up in follow-up rounds.

Finally, PIM computes matchings on per-packet granularity[1]. 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 throughput.

2.1 Two Challenges for dcPIM

As discussed earlier, datacenter networks bear a striking similarity to switching fabrics. However, naïvely using PIM on datacenters can lead to high latency and low network utilization. The latency problem is due to datacenter networks having much larger scales than switch fabrics: using $O(\log n)$ rounds on a datacenter network means that a flow may have to wait for $\sim$20 RTTs for its sender and receiver to match before it can start data transmission! To be able to efficiently realize PIM for datacenters, we need a better tradeoff between number of rounds and utilization guarantees.

Naÿvely using PIM on datacenters can lead to low underutilization due to several reasons related to datacenter networks having RTTs of the order of microseconds. First, even if we run PIM for fewer rounds than $O(\log n)$, computing matchings will still take tens of microseconds. Performing PIM-style sequential matching and data transmission (that is, compute a matching, transmit data between the matched end-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. Second, computing matchings at per-packet granularity will 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.

3 dcPIM Design

dcPIM design comprises of two "phases", a matching phase followed by a data transmission phase, as in PIM. However, it overcomes the challenges of adapting PIM to datacenters using three key ideas. First, using a new theoretical analysis of the PIM protocol, dcPIM demonstrates that for the special case of datacenter networks, it is possible to achieve near-optimal network utilization with a small, constant, number of rounds in the matching phase (§3.1). Second, 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 end-hosts, dcPIM extends PIM's original matching algorithm to match each sender with multiple receivers and vice versa (§3.4).

Four basic aspects of dcPIM design are borrowed from recent work on transport designs that perform packet scheduling and/or admission control [17, 21, 35]. First, upon arrival of a new flow, the sender sends a `notification` packet to the corresponding receiver; this packet may contain information necessary for the receiver to make matching decisions (*e.g.*, size of the flow, deadlines, tenant information, etc.), if such information is available. These notification packets allow the receivers to keep track of the number of outstanding packets to be admitted for each flow. Second, similar to [17, 21, 35], all switches in dcPIM perform per-packet spraying [14]; dcPIM assumes no additional functionality from the network hardware. Third, all control packets in dcPIM are sent at the highest priority. As argued in [17, 21, 35], this implies that the network fabric behaves like a lossless fabric for control packets since existing commodity switches have enough buffer space [1]. Nevertheless, dcPIM employs simple mechanisms to handle extremely rare loss of control packets (Appendix A.2). Finally, if the new flow is very short, the source transmits the flow immediately with second highest priority; this mechanism, similar to [17, 21, 35][2], allows minimizing short flow completion times. For longer flows, dcPIM's mechanism kicks in: the flow is transmitted once the source is matched to the corresponding receiver and once the receiver admits the packets for the flow. 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 similar to PIM (§2): senders and receivers exchange control messages over multiple rounds to match with each other. A minor difference is that in dcPIM, receivers start the round by sending requests since they already get notified about, and keep track of, senders with outstanding data via notification packets. The core difference between dcPIM and PIM is a new tradeoff between number of

[1]in fact, to account for varying packet sizes, many switches schedule packets over the fabric at the granularity of fixed-sized cells.

[2]There is a subtle difference, though: while [17, 21, 35] transmit first few packets of *all* flows immediately, dcPIM does so only for short flows. Network operators can configure which flows are considered short; dcPIM evaluation uses the same parameter as in previous designs: flows of size less than or equal to 1 BDP (using unloaded RTTs) are considered short.

rounds and matching size based on a new theoretical analysis, which we outline next.

Theoretical guarantees with constant number of rounds. Our analysis exploits the sparsity of traffic matrices in datacenter networks: several traffic analysis studies from production datacenters [11, 16, 18, 39, 40] 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 end hosts). For instance, [11] 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).

To exploit such sparse traffic matrices, 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 of the receivers. If PIM computes a matching of expected size $M^{\star} = n/\alpha$, for some $1 \leq \alpha \leq n$, 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 because 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).

.

Proof Outline. We present the formal proof in Appendix A.3. We have to employ a new proof strategy due to two reasons. First, 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 $O(\log n)$). Second, 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.

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. From PIM analysis [10], we get 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. From the PIM analysis [10] and as we also show in Appendix A.3, we also get that after r rounds, the expected number of unresolved requests for a sender u is $\deg(u)/4^r$, where $\deg(u)$ is the number of unresolved requests before round 0 (in the original PIM analysis, $\deg(u) = n$ for each sender). From here on, our proof differs from PIM analysis.

Let $\mathcal{A}$ be the set of active senders after r rounds. Our 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 and since the average degree of the graph is bounded, the size of $\mathcal{A}$ cannot be very large. Specifically, we show that the expected size of $\mathcal{A}$ is $E/4^r$, where E is the total number of edges in the graph. Since our graph is bipartite, E is also equal to $\bar{\delta}\cdot n$. Therefore, the expected size of $\mathcal{A}$ is $\bar{\delta}\cdot n/4^r$. 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 $\log(n)$ rounds is bounded by the size of $\mathcal{A}$. Let $M^{\star} = n/\alpha$ be the matching size computed by PIM after $\log(n)$ rounds; then, we have that the size of matching computed by dcPIM after r rounds is:

$$M_{dcPIM} \geq M^{\star} - |\mathcal{A}| \geq \frac{n}{\alpha} - \frac{\bar{\delta}\cdot n}{4^r} \geq \frac{n}{\alpha}\cdot(1 - \frac{\bar{\delta}\cdot\alpha}{4^r}) \geq M^{\star}\cdot\left(1 - \frac{\bar{\delta}\cdot\alpha}{4^r}\right)$$

Matching size versus utilization guarantees. The above theorem assumes full-bisection bandwidth network topology (similar to recent literature [21, 23, 35]); however, dcPIM does not require such an assumption in its design and implementation. Extending our analysis to oversubscribed topologies is an intriguing open problem (that appears non-trivial). Even for full-bisection bandwidth networks, our discussion so far has focused on size of the matching computed by dcPIM, and how this matching closely approximates a maximal matching. However, once a receiver is matched to a source, it admits packets only from that source. Thus, matchings correspond to network utilization only if the source 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 the entire data transmission phase, hence providing utilization guarantees same as the matching guarantees.

3.2 The Data Transmission Phase

In data transmission phase, each receiver admits packets from the matched sender by sending a per-packet "token" that specifies a flow ID, a sequence number for the admitted packet and cumulative acknowledgements similar to traditional transport protocols.

dcPIM token clocking mechanism. To handle congestion due to oversubscription or due to short flows being transmitted without matching, dcPIM receivers use a "token clocking" mechanism similar to TCP and some recent designs [17]. However, unlike TCP that uses ACK clocking to smooth out the bursts in packet transmissions at the sender, the goals of dcPIM token clocking mechanism is to efficiently handle packet delays in matched flows (due to oversubscription or high-priority short flow packets).

Each receiver maintains a per-flow sliding token window (1 BDP by default). Tokens specify both the flow ID and the admitted data packet sequence number. 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. 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 source are delayed, either due to oversubscription or high-priority short flows, the window may fill up. At this point, the receiver sends one token for every data packet received from the matched source. If the receiver does not receive any data packet and the window is full, the tokens in the window may be resent after a timeout period, within the same data transmission phase or when the receiver matches with the corresponding source in future matching phases.

dcPIM's token clocking mechanism combined with its matching-based design efficiently handles congestion due to oversubscription or high-priority short flows, alleviating the need for sender-side congestion control protocol atop dcPIM: 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. As a result, dcPIM clocking mechanism handles both admission control and reliability on a per-packet granularity.

Sender-side data transmission logic. dcPIM sender-side data transmission logic is simple. 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 second highest priority. Otherwise, the source 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 source to choose (among concurrently active flows) the next packet to transmit, and to set in-network priorities similar to [17, 35].

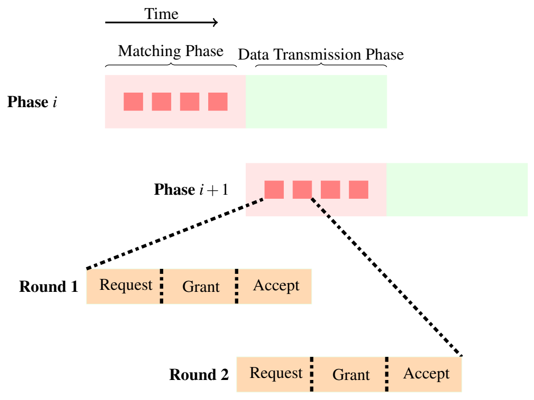


Figure 2: An illustration of dcPIM's pipelining (§3.3)

dcPIM senders discard unused tokens two MTU transmission time after the token is received (for the extension where dcPIM allows each sender to match with k receivers, each token is discarded after $2k$ MTU transmission times). This 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.

3.3 Pipelining Phases

dcPIM exploits the fact that datacenter networks are rarely run at an average load of 100%, providing enough bandwidth to simultaneously send data and control packets. dcPIM, thus, *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 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 three ideas:

Near-optimal matchings using constant number of rounds. First, rather than using $\sim\log(n)$ matching rounds, dcPIM allows computing near-optimal matchings (nearly as good as PIM's matching) using a small constant number of matching rounds r. Network operators can set the value of r based on expected *average* number of concurrent flows—most networks already profile their traffic to compute such metrics; we use $r = 4$ in our evaluation and also present sensitivity analysis of dcPIM for varying values of r.

Embracing PIM's simplicity: fixing the time for each stage. The length of each stage determines the amount of time end-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 some "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 the fixed stage 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 specified time since end hosts unmatched in one round will be able to catch up in the remaining 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.

Pipelining stages across multiple rounds. Each matching round requiring three stages (request, grant, accept) means that c 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 with the request stage at the beginning of the next round (details in §A.1). With such pipelining, 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 source, it admits packets only from that source. That means matchings correspond to network utilization only if the source has enough packets to be admitted by the receiver to "fill up" the entire data transmission phase (§3.1).

Consider, for instance, a leaf-spine datacenter network topology with 100Gbps links; as detailed in §4, for this topology, network RTT is 5.8μs, cRTT $= 5.2\mu$s, and bandwidth-delay product is 72.5KB. For our default values $r = 4, \beta = 1.3$, the length of the data transmission phase will be $(2r + 1)\beta\frac{\text{cRTT}}{2} = 30.42\mu$s. Thus, if the source 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 (5.84μs 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 $(1 - (5.84/30.42)) \approx 0.81$ fraction of the data transmission phase, potentially missing the opportunity to send or receive data from other idle end 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) one channel at a time. Each channel operates using $1/k$ of the link bandwidth. 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 $\sim$95% 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 when each receiver is matched with at most one sender ($k = 1$, §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 indicate the maximum number of channels it can allocate to that sender in the request stage (this is equal to the minimum of number of unmatched channels and 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 deducts the number of bytes from the outstanding bytes from that sender to account for the number of bytes that will be sent over the matched channel during the data transmission phase.

The current dcPIM design divides the link bandwidth equally across all k channels; this could be extended to have receivers allocate non-uniform bandwidth to each of the k matched senders. It is an intriguing algorithmic problem to design an algorithm that performs such non-uniform allocation while maintaining dcPIM's theoretical guarantees on network utilization.

The Data Transmission Phase. All k channels of a receiver (or sender) share its available bandwidth in the data transmission phase, each sending and receiving data at $\frac{1}{k}$ of the link rate. 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 dcPIM parameters: r, β and k

dcPIM design has just 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. Our evaluation (§4) shows that 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 data transmission phase, which lasts for $\sim r$ RTTs. When k is equal to r, k flows can concurrently transmit for the entire data transmission phase. Thus, this is the sweet sport that provides significant utilization gains without considerably hurting FCT.

We perform sensitivity analysis of dcPIM performance against these parameters in §4.

3.6 Asynchronous Design & Optimizations

We have already described most of the interesting aspects of dcPIM design in the previous subsections. In this subsection, we outline a few more pragmatic and implementation details.

Asynchronous Design. Our design and implementation of dcPIM does not require perfect synchronization across the end hosts. Each end 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 moden NICs support PTP. Modern protocols can achieve time synchronization that is much finer than a single MTU-sized packet transmission time [33, 38]. Moreover, as discussed earlier, randomization and multi-round nature of PIM ensures that end-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 find near-optimal matchings (close to that of PIM's matchings), that is, 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 ($\leq$ 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 [17] and Homa [35] that approximate 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 extreme incast. If packets from short flows, which are sent without matching, are dropped (*e.g.*, many sources have short flows at the same time), the outstanding packets in the flow need to be explicitly admitted [23]. That is, the flow will participate in the next matching phase and packets need to be admitted by the receiver after matching. Intuitively, (rare) packet drops in dcPIM indicate high congestion; 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.

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:

Take-away 1—*dcPIM consistently achieves good short flow latency and network utilization*. Across a variety of settings that mix-and-match three datacenter workloads, three traffic patters, three datacenter network topologies, varying network loads, and varying network bandwidths, dcPIM consistently achieves average and tail short flow latency similar or better than state-of-the-art protocols while sustaining higher loads. We will also demonstrate, via microbenchmarks, that

Topologies	- Two-tier leaf-spine, commonly used in the literature [9, 17, 35]: 4 spines, 9 racks each with 16 end-hosts - Fat-Tree: Three-tier 432 end-hosts - Oversubscribed: same as two-tier leaf-spine but with 2:1 oversubscription ratio.
Workloads	- IMC10 [17], Web Search [9], and Data Mining [9]
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 [27]
Evaluated Protocols	- Homa Aeolus [23, 35] and NDP [21]: Receiver-driven transport protocols - HPCC [7]: Rate-control using precise link load information from the network (through INT)
Evaluated Metrics	- Slow-down: 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 to the maximum possible network throughput)
Link Properties	- 100Gbps access links for end-hosts, 200ns transmission 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 [9, 17]) (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.

dcPIM's theoretical foundation allows it to achieve good utilization even under adverse workloads.

Take-away 2—*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 trade-off: in the absence of short flows, the queueing induced by long flows is negligible since matched flows constitute a permutation traffic matrix that can be sustained by full bisection bandwidth networks using packet spraying. 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 bounds ensures that maximum buffering is exactly 1BDP, precisely what is needed to keep the downlink busy for the next 1RTT. Thus, the main take-away is not that dcPIM achieves near-optimal average and tail latencies, but rather that dcPIM does so without giving up on network utilization.

4.1 dcPIM Simulation Results

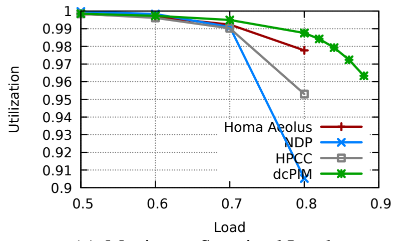
We incorporate dcPIM within the pFabric simulator [9]. The workloads, topologies, performance metrics, and protocols used in our simulations are summarized in Table 1. For all evaluated protocols, we use their respective simulators but ensure consistency in topology and workload settings.

Default Setup. Unless stated otherwise, we use the standard setup from prior work [9, 21, 23, 35]: 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 [17, 21, 23, 35], 1 BDP as short flow size threshold. For other protocols, we set the parameters suggested in respective papers. See Table 1 for details.

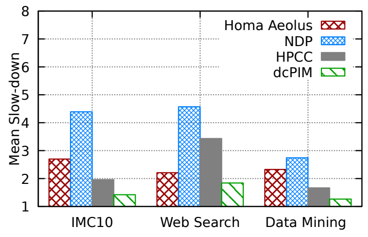
dcPIM can maintain near-optimal short flow latency without giving up on utilization. Figure 3 shows that existing protocols either achieve near-optimal latency for short flows or high utilization, but not both. dcPIM, on the other hand, is able to simultaneously achieve both. We provide intuitive reasons for each protocol individually.

Homa Aeolus achieves the best network utilization among existing protocols, coming closest to dcPIM (Figure 3(a)), and good average slow-downs across all flows (Figure 3(b)). However, it does so by trading off short flow latency (Figure 3(c), 3(d), 3(e)): we observe short flow slow-down 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 assumes infinitely large buffers in switches, 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 extremely 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); however, Homa suffers from significantly degraded network utilization. Homa Aeolus makes the better tradeoff when compared to Homa: give up a bit on short flow latency to achieve much better utilization.

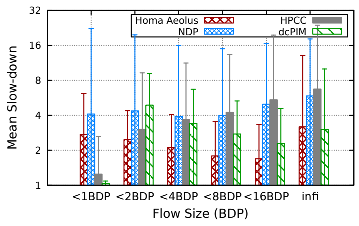
NDP suffers from high short flow latency as well as poor network utilization. We observe short flow slow-down of 2.5–4 on an average, and 12.5–22.3 at tail across various workloads. The reason is two folds: (1) similar to Homa Aeolus, NDP proactively drops packets to maintain low queue occupancy; and (2) NDP does not use prioritization. NDP's proactive dropping of packets combined with lack of Homa Aeolus kind of mechanism to ensure that retransmitted pack-



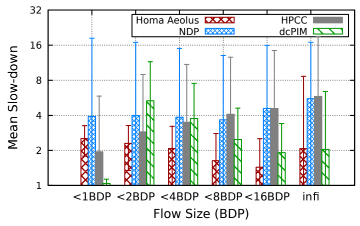
(a) Maximum Sustained Load



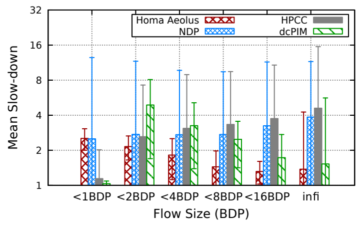
(b) Mean Slow-down



(c) Slow-down across flow sizes (IMC10)



(d) Slow-down across flow sizes (Web Search)



(e) Slow-down 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 slow-down 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 slow-down broken down by flow size (x-axis labels).

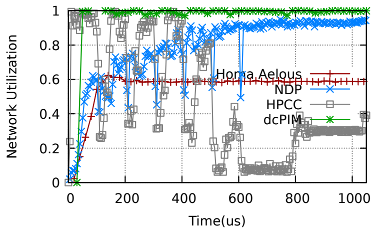
ets are not dropped again also results in poor utilization.

HPCC is an interesting case. HPCC performs aggressive rate control to ensure minimal queue occupancy at each switch in the network. This allows HPCC to achieve very low latency for short flows—we observe average slow-down of 1.2–1.9 on an average and tail slow-down of 2–5.9 across various workloads. This is because senders transmit short flows at 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.

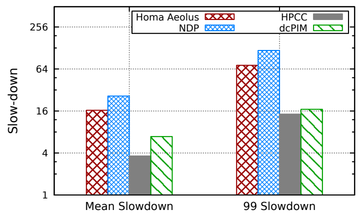
dcPIM is able to achieve near-optimal average and tail latency for short flows, while maintaining high network utilization. For example, across workloads, we observe average short flow slow-down of 1.030–1.039 (as much as 2.7×, 4× and 1.9× better than Homa Aeolus, NDP and HPCC, respectively) and tail short flow slow-down of 1.07–1.12× (as much as 5.8×, 20.9× and 5.2× 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 workload, higher than all other evaluated protocols. We observe similar results for the Web Search and Data Mining workloads: dcPIM can sustain 0.84 and 0.7 load, respectively, much higher than other protocols.

How is dcPIM able to maintain high network utilization? We already summarized the reasons in the beginning of the evaluation section (Take-away 2), 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 bounds, dcPIM would achieve much better network utilization than the theoretical bounds. 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 [11, 18, 39]). 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.

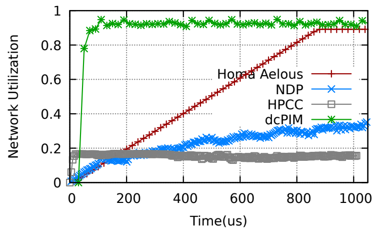
dcPIM maintains low latency for short flows without impacting network utilization by making a specific tradeoff: latency of medium-sized flows since the time taken to match flows before they can be transmitted incurs the most overhead for flows that are not too short and are not too long. We believe 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



(a) dcPIM robustness in bursty workloads



(b) dcPIM worst case: all flows of size BDP+1



(c) dcPIM bad case: dense traffic matrix ($\bar{\delta} = 144$)

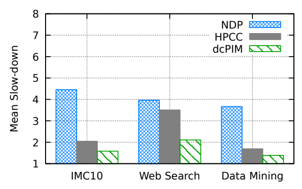
Figure 4: Microscopic view into dcPIM performance. (left) even under extremely bursty (all-to-all traffic pattern combined with periodic incast) workloads, dcPIM is able to maintain high network utilization. (center) for the hypothetical case of all flows of size BDP+1 (worst-case for dcPIM, §3.4), dcPIM has slightly higher latency than HPCC; and (right) dcPIM achieve high utilization when even average number of concurrent flows is not too small. Results are discussed in §4.1.

flow completion time for medium size flows. 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).

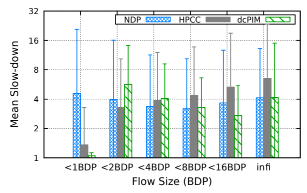
Microscopic view into dcPIM benefits. Large-scale simulations often hide how protocols handle short-term traffic bursts. In our next experiment, we setup a microbenchmark: on the same two-tier leaf-spine topology, we have 16 senders in the same ToR send an all-to-all traffic to 16 receivers in the other 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 and Homa Aeolus stumble under such a workload. For HPCC, we observe PFC triggering frequently; for Homa Aeolus, we observe poor “matching”—multiple receivers repeatedly sending 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. NDP, by performing fair sharing of grants across all senders, converges to high utilization but takes almost 600μs to converge. 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. We do not show short flow latency results since NDP, HPCC and Homa Aeolus do not finish all the short flows during the microbenchmark period.

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 websearch 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) 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.

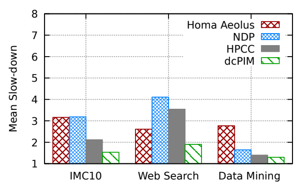
What if the sparse traffic matrix assumption does not hold? As discussed earlier, datacenter traffic analysis studies have shown that the *average* incast ratios in production datacenters are small. However, it may happen that incast ratios may be large over a subset of the datacenter over a short period 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 $= 144 \times 144$), 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 800μs to converge. dcPIM performance degrades compared to previous results, but it still achieves ~89% 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 39 in any matching phase. Digging deeper into this result, we found that dcPIM high performance is because



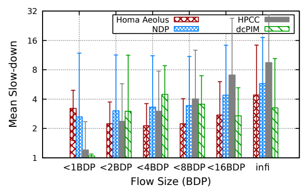
(a) Oversubscribed Slow-down



(b) Oversubscribed Slow-down per Flow Size



(c) Fat-Tree Slow-down



(d) Fat-Tree Slow-down 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 432-node FatTree topology. * we were unable compare against Homa Aeolus in oversubscribed topologies. See §4.1 for detailed discussion.

(1) over multiple phases, dcPIM is able to converge to larger matching sizes (as is seen during the first $\sim 100\mu$s in the figure); and (2) dcPIM analysis does not take into account the multi-channel approach from §3.4, which significantly improves the size of matching computed by dcPIM. We believe extending dcPIM analysis to incorporate multiple channels is an interesting theoretical problem.

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 1:2. We generate all-to-all traffic traces from all three workloads for a network load of 0.5. 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. We spent several weeks trying to overcome these issues in respective simulators, but the full-bisection bandwidth assumption is deeply embedded in all the simulators. For both oversubscribed and FatTree topologies, the observed results have the same trend as in Figure 3, for precisely the same reasons as outlined earlier.

We have also evaluated dcPIM over several additional workloads, including mix of all-to-all traffic with bursty incast traffic [27]. The results are shown in Figure 8 in Appendix A.4. HPCC and NDP exhibit the same behavior as earlier; Homa Aeolus suffers from low utilization when compared to earlier results due to larger number of packet drops. dcPIM continues to achieve low average and tail latency, while sustaining higher utilization than other protocols.

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 0.56 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 20–23% higher sustainable load with two rounds (additional rounds result in diminishing returns); improved utilization also results in improved average latency[3]. 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 not utilization for dcPIM.

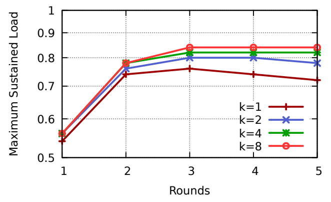
4.2 dcPIM Implementation Results

dcPIM prototype is implemented in Linux using DPDK [3] using $\sim$3K lines of C code. Our prototype uses one core to pace packets, and one to execute dcPIM's logic. We use CloudLab [2] to evaluate dcPIM implementation over a 32-server two-tier leaf-spine topology (10Gbps links and $\sim 8\mu$s RTT). As discussed in §3.6, dcPIM can operate upon loosely synchronized end host clocks (*e.g.*, using time synchronization protocols like PTP [13]). Our testbed, however, does not support PTP. Thus, our results are worst-case results: we expect dcPIM to perform better in datacenters that typically use time synchronization protocols.

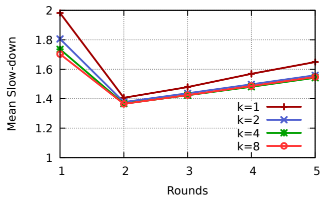
Figure 7 shows the average and tail slow-down for dcPIM, DCTCP and TCP Cubic[4] over CloudLab testbed for the same all-to-all traffic pattern as earlier, but using load 0.5. The results are as expected: for short flows, dcPIM achieves 21×/43× and 58×/76× better average and tail latencies

[3]Note that short flow tail latency is not shown since dcPIM parameters do not impact short flow latency—they are always prioritized.

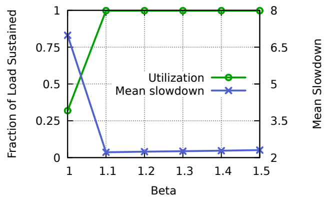
[4]We could not run Homa Aeolus implementation since it only supports incast workloads. Similarly, Homa DPDK implementation only supports client-server model, and the kernel implementation does not support RPC sizes larger than 1MB [37]. However, as a reference point, Homa's Github page [4] suggests that it improves TCP latency by a factor of 10 over RPC traffic.



(a) Maximum Sustained Load (r,k)



(b) Mean Slow-down (r,k)



(c) Sustained Load & Mean Slow-down (β)

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

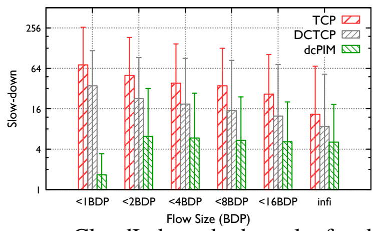


Figure 7: 32-server CloudLab testbed results for dcPIM, DCTCP and TCP cubic: mean and tail slow-down (error bars show 99%) across different flow sizes. y-axis is log-scale. See §4.2.

than DCTCP/TCP respectively. As flow sizes increase, DCTCP/TCP achieves results more comparable to dcPIM (within 2.84×/3.74× average latency for long flows respectively). We have also evaluated dcPIM scalability using 8, 16 and 32 servers over the same setup. The results are presented in Figure 9 in the appendix. We observe that dcPIM scales well as the topology size increases, consistently achieving low latency. We do observe that, as the topology size increases, the slow-down for dcPIM increases slightly. This is an artifact of our testbed not supporting PTP (larger clock drifts at larger scales). Despite such worst-case condition, dcPIM is able to provide consistent performance by embracing imperfections similar to PIM.

5 Related Work

There is a large and active body of research on datacenter transport protocols [6, 7, 8, 12, 20, 22, 36, 41, 47, 48]; it would be a futile attempt to list all protocols. 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 [38]. 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, such a design not only leads to scalability issues, but also 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 latency.

The use of matching mechanisms has a long history in switch scheduling [10, 24, 25, 30, 31, 32, 42, 43, 46], 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 [28]; other protocols (*e.g.*, IRRM [32] and iSLIP [30]) 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. Nevertheless, we believe dcPIM is only the first step in exploring the many possible facets of switch scheduling mechanisms on datacenter networks.

6 Conclusion

Modern datacenter networks bear a striking similarity to switching fabrics. Motivated by bridging the gap between switch and datacenter scheduling, we have presented datacenter Parallel Iterative Matching (dcPIM), a transport design that realizes the classical Parallel Iterative Matching protocol from switch scheduling literature on datacenter networks. Using theoretical analysis and evaluation, we have shown that dcPIM can achieve near-optimal latency, while consistently maintaining high network utilization. We believe that dcPIM is only the first step in bridging the gap between the scheduling mechanisms in switching fabrics and datacenters.

References

[1] CISCO: Per packet load balancing. http://www.cisco.com/en/US/docs/ios/12_0s/feature/guide/pplb.html.

[2] Cloudlab. https://www.cloudlab.us/.

[3] *Data Plane Development Kit*.

[4] Homa Kernel Module. https://github.com/PlatformLab/HomaModule.

[5] M. Al-Fares, A. Loukissas, and A. Vahdat. A scalable, commodity data center network architecture. In *SIGCOMM*, 2008.

[6] M. Alizadeh, T. Edsall, S. Dharmapurikar, R. Vaidyanathan, K. Chu, A. Fingerhut, V. T. Lam, F. Matus, R. Pan, N. Yadav, et al. Conga: Distributed congestion-aware load balancing for datacenters. In *SIGCOMM*, 2014.

[7] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan. Data Center TCP (DCTCP). In *SIGCOMM*, 2010.

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

[9] M. Alizadeh, S. Yang, M. Sharif, S. Katti, N. McKeown, B. Prabhakar, and S. Shenker. pFabric: Minimal Near-optimal Datacenter Transport. In *ACM SIGCOMM*, 2013.

[10] T. E. Anderson, S. S. Owicki, J. B. Saxe, and C. P. Thacker. High-speed switch scheduling for local-area networks. *ACM Transactions on Computer Systems (TOCS)*, 11(4):319–352, 1993.

[11] T. Benson, A. Akella, and D. A. Maltz. Network traffic characteristics of data centers in the wild. In *IMC*, 2010.

[12] I. Cho, K. Jang, and D. Han. Credit-scheduled delay-bounded congestion control for datacenters. In *SIGCOMM*. ACM, 2017.

[13] R. Cochran, C. Marinescu, and C. Riesch. Synchronizing the linux system time to a ptp hardware clock. 2011.

[14] A. Dixit, P. Prakash, Y. C. Hu, and R. R. Kompella. On the impact of packet spraying in data center networks. In *Proc. of IEEE INFOCOM*, 2013.

[15] N. Farrington, E. Rubow, and A. Vahdat. Data center switch architecture in the age of merchant silicon. In *2009 17th IEEE Symposium on High Performance Interconnects*, pages 93–102. IEEE, 2009.

[16] D. Firestone, A. Putnam, S. Mundkur, D. Chiou, A. Dabagh, M. Andrewartha, H. Angepat, V. Bhanu, A. Caulfield, E. Chung, et al. Azure accelerated networking: Smartnics in the public cloud. In *NSDI*, 2018.

[17] P. X. Gao, A. Narayan, G. Kumar, R. Agarwal, S. Ratnasamy, and S. Shenker. phost: Distributed near-optimal datacenter transport over commodity network fabric. In *CoNEXT*, 2015.

[18] M. Ghobadi, R. Mahajan, A. Phanishayee, N. Devanur, J. Kulkarni, G. Ranade, P.-A. Blanche, H. Rastegarfar, M. Glick, and D. Kilper. Projector: Agile reconfigurable data center interconnect. In *SIGCOMM*, 2016.

[19] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta. Vl2: a scalable and flexible data center network. In *SIGCOMM*, 2009.

[20] M. P. Grosvenor, M. Schwarzkopf, I. Gog, R. N. Watson, A. W. Moore, S. Hand, and J. Crowcroft. Queues don't matter when you can jump them! In *NSDI*, 2015.

[21] M. Handley, C. Raiciu, A. Agache, A. Voinescu, A. Moore, G. Antichi, and M. Wojcik. Re-architecting datacenter networks and stacks for low latency and high performance. In *ACM SIGCOMM*, 2017.

[22] C.-Y. Hong, M. Caesar, and P. Godfrey. Finishing flows quickly with preemptive scheduling. In *ACM SIGCOMM*, 2012.

[23] S. Hu, W. Bai, G. Zeng, Z. Wang, B. Qiao, K. Chen, K. Tan, and Y. Wang. Aeolus: A building block for proactive transport in datacenters. In *SIGCOMM '20*, 2020.

[24] A. Israeli and A. Itai. A fast and simple randomized parallel algorithm for maximal matching. *Information Processing Letters*, 22(2):77–80, 1986.

[25] T. Javidi, R. Magill, and T. Hrabik. A high-throughput scheduling algorithm for a buffered crossbar switch fabric. In *ICC*, 2001.

[26] R. M. Karp, U. V. Vazirani, and V. V. Vazirani. An optimal algorithm for on-line bipartite matching. In *STOC*, 1990.

[27] Y. Li, R. Miao, H. H. Liu, Y. Zhuang, F. Feng, L. Tang, Z. Cao, M. Zhang, F. Kelly, M. Alizadeh, and M. Yu. Hpcc: High precision congestion control. In *SIGCOMM '19*, 2019.

[28] Z. Lotker, B. Patt-Shamir, and S. Pettie. Improved distributed approximate matching. *Journal of the ACM (JACM)*, 62(5):38, 2015.

[29] M. A. Marsan, A. Bianco, P. Giaccone, E. Leonardi, and F. Neri. Multicast traffic in input-queued switches: optimal scheduling and maximum throughput. *IEEE/ACM Transactions on Networking*, 11(3):465–477, 2003.

[30] N. McKeown. The iSLIP scheduling algorithm for input-queued switches. *IEEE/ACM Trans. Netw.*, 7(2):188–201, Apr. 1999.

[31] N. McKeown, A. Mekkittikul, V. Anantharam, and J. Walrand. Achieving 100% throughput in an input-queued switch. *IEEE Transactions on Communications*, 47(8):1260–1267, 1999.

[32] N. McKeown, P. Varaiya, and J. Walrand. Scheduling cells in an input-queued switch. *Electronics Letters*, 29(25):2174–2175, 1993.

[33] D. L. Mills. Network time protocol (ntp), 1985.

[34] R. Mittal, N. Dukkipati, E. Blem, H. Wassel, M. Ghobadi, A. Vahdat, Y. Wang, D. Wetherall, D. Zats, et al. Timely: Rtt-based congestion control for the datacenter. In *SIGCOMM*, 2015.

[35] B. Montazeri, Y. Li, M. Alizadeh, and J. Ousterhout. Homa: A receiver-driven low-latency transport protocol using network priorities. In *SIGCOMM*, 2018.

[36] A. Munir, G. Baig, S. M. Irteza, I. A. Qazi, A. X. Liu, and F. R. Dogar. Friends, not foes: synthesizing existing transport strategies for data center networks. In *ACM SIGCOMM*, 2014.

[37] J. Ousterhout. A linux kernel implementation of the homa transport protocol. In *USENIX ATC 21*, 2021.

[38] J. Perry, A. Ousterhout, H. Balakrishnan, D. Shah, and H. Fugal. Fastpass: A centralized zero-queue datacenter network. In *SIGCOMM*, 2015.

[39] A. Roy, H. Zeng, J. Bagga, G. Porter, and A. C. Snoeren. Inside the social network's (datacenter) network. In *SIGCOMM*, 2015.

[40] A. Saeed, N. Dukkipati, V. Valancius, C. Contavalli, A. Vahdat, et al. Carousel: Scalable traffic shaping at end hosts. In *SIGCOMM*, 2017.

[41] A. Saeed, V. Gupta, P. Goyal, M. Sharif, R. Pan, M. Ammar, E. Zegura, K. Jang, M. Alizadeh, A. Kabbani, and A. Vahdat. Annulus: A dual congestion control loop for datacenter and wan traffic aggregates. In *SIGCOMM*, 2020.

[42] D. Shah, P. Giaccone, and B. Prabhakar. Efficient randomized algorithms for input-queued switch scheduling. *Micro*, 2002.

[43] D. Shah and D. Wischik. Optimal scheduling algorithms for input-queued switches. In *INFOCOM*, 2006.

[44] A. Singh, J. Ong, A. Agarwal, G. Anderson, A. Armistead, R. Bannon, S. Boving, G. Desai, B. Felderman, P. Germano, et al. Jupiter rising: A decade of clos topologies and centralized control in google's datacenter network. In *SIGCOMM*, 2015.

[45] R. E. Tarjan. *Data structures and network algorithms*, volume 44. Siam, 1983.

[46] G. Varghese. *Network Algorithmics: an interdisciplinary approach to designing fast networked devices*. Morgan Kaufmann, 2005.

[47] Y. Zhu, H. Eran, D. Firestone, C. Guo, M. Lipshteyn, Y. Liron, J. Padhye, S. Raindel, M. H. Yahia, and M. Zhang. Congestion control for large-scale rdma deployments. In *SIGCOMM*, 2015.

[48] N. Zilberman, G. Bracha, and G. Schzukin. Stardust: Divide and conquer in the data center network. In *NSDI*, 2019.

A Appendix

A.1 Pipelining Stages across 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 $\beta\frac{\text{cRTT}}{2}$ time, the total length of the matching and consequently the data transmission phases turns out to be $3r\beta\frac{\text{cRTT}}{2}$. We observe that it is possible to further reduce this length by pipelining the accept stage at the end of each with the request stage at the beginning of the next round.

Specifically, the set of end hosts that participate in the accept and in the request stages are completely independent: 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 (as shown in Figure 2). Note that when an unmatched sender receives an `accept` packet, it discards all other `request` 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 $(2r+1)\beta\frac{\text{cRTT}}{2}$.

A.2 Handling (rare) Control Packet Drops

Recall that dcPIM sends all control packets at the highest priority. Thus, even with commodity switches, the network fabric behaves almost like a lossless fabric for control packets. Nevertheless, dcPIM is tolerant to the rare loss of control packets. If `grant` or `request` packets are lost in one round, their corresponding senders and receiver 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 randomization. Dropped `token` packets are treated as dropped data packets and dcPIM recovers from their loss using the token clocking mechanism described in §3.2. Finally, dcPIM uses acknowledgements and timeouts to recover from the loss of `notification` packets, which announce the arrival of a new flow: the receiver simply sends an acknowledgment for each `notification` packet to the sender, so that the sender can retransmit them if they are not acknowledged within a configured period of time.

A.3 Proof for 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.

Specifically, let $Q_{u,r}$ be a random variable representing the number of unresolved requests for sender u after r rounds. First, we show that for any q, if after round $r-1$, $Q_{u,r-1} = q$, then $\mathbb{E}[Q_{u,r}|Q_{u,r-1} = q]$, that is, the expected number of remaining unresolved requests for sender u after round r, is at most $\frac{q}{4}$. In round r, u receives q unresolved requests from different receivers. Those 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 request 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 sender S 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, $\mathbb{E}[Q_{u,r}|Q_{u,r-1} = q] \leq \frac{q}{4}$.

Using Bayes rule and linearity of expectation, we have:

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

That is, $\mathbb{E}[Q_{u,r}] \leq \frac{\mathbb{E}[Q_{u,r-1}]}{4}$. Therefore:

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

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 $\deg(u)/4^r$ (in the original PIM analysis, $\deg(u) = n$ for each sender). Therefore

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

From here on, our proof differs from PIM analysis. Let $I_{u,r}$ be an indicator random variable that is one if $Q_{u,r} \geq 1$ and zero otherwise. That is, $\mathbb{E}[I_{u,r}] = Pr(I_{u,r} = 1) = Pr(Q_{u,r} \geq 1)$. Markov inequality says for a nonnegative random variable X and $a > 0$, $Pr(X \geq a) \leq \frac{\mathbb{E}[X]}{a}$. 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

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

and therefore:

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

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 $Q_{u,r} \geq 1$. Thus, the expected number of active senders after r rounds is

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

Since our graph is bipartite, the sum of the degrees of senders is equal to the number of edges (requests), m:

$$\Sigma_{u \in U} deg(u) = m$$

Similarly, the sum of the degrees of receivers is also equal to m. That is, $m = \bar{\delta} \cdot n$, where $\bar{\delta}$ is the average degree of the receivers.As such, we have:

$$\mathbb{E}[|\mathcal{A}|] \leq \bar{\delta} \cdot n / 4^r$$

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 the size of $\mathcal{A}$. Specifically, Let M_{PIM} be the size of the matching computed by PIM after $\log(n)$ rounds, and let $M^\star = \mathbb{E}[M_{PIM}]$. If $M^\star = n/\alpha$ then, we have that the size of matching computed by dcPIM after r rounds is:

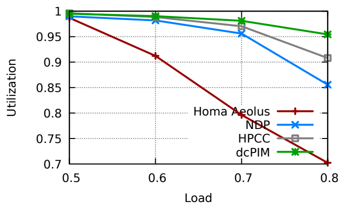
$$\begin{aligned}
\mathbb{E}[M_{dcPIM}] &\geq \mathbb{E}[M_{PIM}] - \mathbb{E}[|\mathcal{A}|] \\
&= M^\star - \mathbb{E}[|\mathcal{A}|] \\
&\geq \frac{n}{\alpha} - \frac{\bar{\delta} \cdot n}{4^r} \\
&= \frac{n}{\alpha} \cdot \left(1 - \frac{\bar{\delta} \cdot \alpha}{4^r}\right) \\
&= M^\star \cdot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

A.4 Additional Evaluation Results

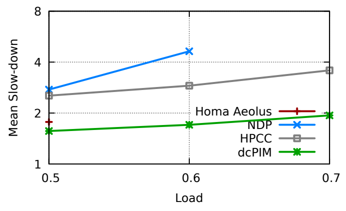
Bursty Traffic. To evaluate dcPIM's performance in the presence of traffic bursts, we generate a bursty traffic pattern by adding incast traffic to our all-to-all traffic pattern. Similar to prior work [34], we generate 50:1 incast by randomly selecting 50 senders and one receiver, and sending 64KB RPC's from senders to the one receiver. The overall incast traffic load remains within 2% of the total network capacity. We then evaluate dcPIM's performance for the bursty traffic pattern generated from IMC10, Web Search, and Data Mining workloads over the default topology and with default protocol parameters. We observe similar trends across all workloads.

As shown in figure 8(a), dcPIM sustains a higher load than other protocols. Overall, the trends are very similar to the all-to-all traffic pattern results except for Homa Aeolus, which has a significantly lower utilization. This is because, unlike all-to-all traffic, Homa Aeolus experiences packet drops for bursty traffic. The authors of Aeolus recommended using an overcommitment level of 1 (that is, receiver always sends one grant per MTU-transmission time) for these experiments as the simulator does not work with higher overcommitment levels used in Homa in such scenarios. In contrast, for experiments with all-to-all traffic, we could run Homa Aeolus with an overcommitment level of 7. That is why Homa Aeolus experiences much lower utilization for bursty traffic.

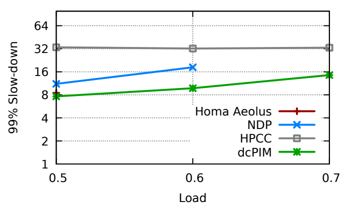
Figure 8(b) and 8(c) depict the mean and tail slow-down of dcPIM and other protocols for the bursty traffic pattern. Note we have not plotted slow-down results for Homa Aeolus beyond 0.5 load and NDP beyond 0.6 load, as they cannot sustain higher loads. We observe that dcPIM achieves better or similar slow-down both on an average and at tail. This is because the size of incast flows (64 KB) is less than one BDP (75 KB). As such, not only will they be transmitted immediately without having to go through matchings, but they will also be prioritized in the network. Moreover, dcPIM maintains short queues in the network by using matchings. Therefore, many of the incast flows will not be dropped, and those that are dropped are prioritized in the FCT-optimizing round of the next matching phase.



(a) Max Sustained Load

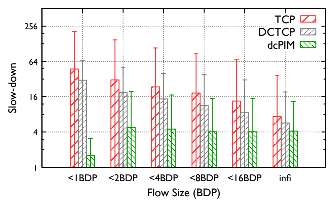


(b) Mean Slow-down vs. load

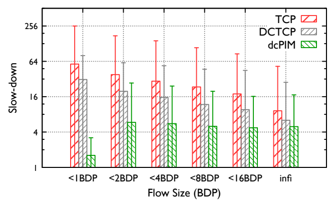


(c) 99% Slow-down vs. load

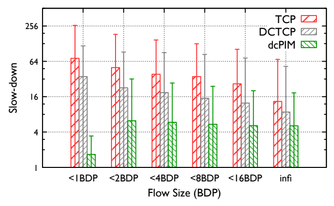
Figure 8: dcPIM's performance for the bursty traffic workload; see Appendix A.4 for a more detailed discussion. dcPIM sustains higher loads (Figure 8(a)) and has similar or better mean and tail slow-down compared to other protocols (Figure 8(b) and 8(c)).



(a) CloudLab 8-node testbed



(b) CloudLab 16-node testbed



(c) CloudLab 32-node testbed

Figure 9: Scalability results for CloudLab testbed implementation. The figure shows mean and 99% (error bars) slow-down for dcPIM, DCTCP and TCP Cubic over 8-node, 16-node and 32-node testbed for the same setup as in §4.2. Results are discussed in §4.2.