# Friends, not Foes – Synthesizing Existing Transport Strategies for Data Center Networks

Ali Munir[1]*, Ghufran Baig[2], Syed M. Irteza[2], Ihsan A. Qazi[2],
Alex X. Liu[1], Fahad R. Dogar[3]
[1]Michigan State University, [2]LUMS, [3]Microsoft Research
{munirali, alexliu}@cse.msu.edu, {ghufran.baig, syed.irteza, ihsan.qazi}@lums.edu.pk,
fdogar@microsoft.com

## ABSTRACT

Many data center transports have been proposed in recent times (e.g., DCTCP, PDQ, pFabric, etc). Contrary to the common perception that they are *competitors* (i.e., protocol A vs. protocol B), we claim that the underlying strategies used in these protocols are, in fact, *complementary*. Based on this insight, we design PASE, a transport framework that *synthesizes* existing transport strategies, namely, self-adjusting endpoints (used in TCP style protocols), in-network prioritization (used in pFabric), and arbitration (used in PDQ). PASE is deployment friendly: it does not require any changes to the network fabric; yet, its performance is comparable to, or better than, the state-of-the-art protocols that require changes to network elements (e.g., pFabric). We evaluate PASE using simulations and testbed experiments. Our results show that PASE performs well for a wide range of application workloads and network settings.

**Categories and Subject Descriptors:** C.2.2 [Computer-Communication Networks]: Network Protocols
**Keywords:** datacenter; transport; scheduling

## 1. INTRODUCTION

Popular data center applications (e.g., search) have many distributed components that interact via the internal data center network. Network delays, therefore, inflate application response times which, in turn, affects user satisfaction. Thus, several recent data center transport proposals focus on providing low latency for user-facing services. These proposals optimize for specific application goals, such as minimizing flow completion times (FCT) or meeting flow deadlines, and use a variety of underlying techniques to achieve their objective, from the more traditional TCP-style approaches (e.g., DCTCP [11]) to those that use an explicit rate control protocol (e.g., PDQ [18]) or a new prioritized network fabric (e.g., pFabric [12]).

All these techniques have their own strengths and weaknesses: some provide good performance, but require changes to the network fabric [12, 18, 24], while others are deployment friendly, but have inferior performance [22, 23]. In this paper, we take a clean slate approach towards designing a data center transport that provides good performance while also being deployment friendly. To this end, we take a step back and ask the following question: *how can we design a data center transport from scratch while leveraging the insights offered by existing proposals?*

As a first step towards answering this question, we distill the three underlying strategies used in data center transports: *i) self adjusting endpoints*, where senders independently make their rate increase/decrease decision based on the observed network conditions (used in DCTCP [11] style protocols), *ii) arbitration*, where a common network entity (e.g., a switch) allocates rates to each flow (used in $D^3$ [24], PDQ [18], etc), and *iii) in-network prioritization*, where switches schedule and drop packets based on their priority (used in pFabric [12]).

We observe that each strategy has its own role. It works best when it is fulfilling its role and encounters problems otherwise. Unfortunately, existing protocols only use *one* of the above strategies and try to make it work under *all* scenarios. In this paper, we make a case that these transport strategies should be used *together* as they nicely complement each other. For example, approaches that rely on arbitration alone have high flow switching overhead because flows need to be explicitly paused and unpaused [18]. With in-network prioritization, switching from a high priority flow to the next is seamless [12]. Conversely, arbitration can also help in-network prioritization approaches. For example, in-network prioritization mechanisms typically need to support a large number of priority levels whereas existing switches only have a limited number of priority queues. An arbitrator can address this problem by *dynamically* changing the mapping of flows to queues — flows whose turn is far away can all be mapped to the lowest priority queue while flows whose turn is about to come can be mapped to the high priority queues.

To demonstrate the benefits of using these strategies together, we design PASE, a transport framework for private data center environments. PASE employs distributed arbitrators that decide the share (priority) of a flow given other flows in the system. A TCP-like end-host transport uses this information for its rate control and loss recovery. Within the network, PASE uses existing priority queues to prioritize the scheduling of packets over network links. This appropriate

*Part of the work was done while the author was an intern at Microsoft Research.


*SIGCOMM'14,* August 17–22, 2014, Chicago, IL, USA.

http://dx.doi.org/10.1145/2619239.2626305.

division of responsibilities among the three strategies makes PASE outperform state-of-the-art transport protocols while also being *deployment friendly* i.e., no changes to the network fabric are required.

A key aspect of PASE is a scalable control plane for arbitration. For every link in the data center topology, a dedicated arbitrator is responsible for arbitration. This functionality can be implemented at the end-hosts themselves (e.g., for their own links to the switch) or on dedicated nodes within the data center. We exploit the typical tree based data center topology features to make the arbitration decisions in a bottom up fashion, starting from the endpoints and going up to the core. This has several performance and scalability benefits. First, for intra-rack communication, which can constitute a sizeable share of data center traffic [11], only the source and destination are involved, obviating the need to communicate with any other entity. Second, lower level arbitrators (those closer to the leaf nodes) can do *early pruning* by discarding flows that are unlikely to become part of the top priority queue. Third, higher level arbitrators (those closer to the root) can *delegate* their arbitration decisions to lower level arbitrators. Both early pruning and delegation reduce the arbitration overhead (at the cost of potentially less accurate decisions).

The outcome of arbitration is the *priority queue* and *reference rate* for the flow – this information is used by PASE's end-host transport for rate control and loss recovery. Compared to traditional transport protocols (e.g., TCP/DCTCP), our rate control is more *guided*. For example, instead of slow start, the transport uses the reference rate as its starting point. However, loss recovery in PASE is more challenging because packets can be delayed in a lower priority queue for a long time, which may trigger spurious timeouts if we use today's timeout mechanisms. Thus, for lower priority flows, instead of retransmitting the data packet, PASE uses small *probe* packets which help in determining whether the packet was lost, or waiting in a lower priority queue.

We implement PASE on a small testbed and in the ns2 simulator [6]. We compare PASE with the best performing transport protocol (pFabric [12]) as well as deployment friendly options (D$^2$TCP [23]). Compared to deployment friendly options, PASE improves the average FCT (AFCT) by 40% to 60% for various scenarios. PASE also performs within 6% of pFabric in scenarios where pFabric is close to optimal while in other scenarios (all-to-all traffic pattern or under high network load), PASE outperforms pFabric by up to 85% both in terms of the AFCT as well as the $99^{th}$ percentile FCT.

This paper makes the following key contributions.

- We distill the underlying strategies used in data center transport protocols and highlight their strengths and weaknesses.
- We design PASE, a data center transport framework which synthesizes existing transport strategies. PASE includes two new components: a scalable arbitration control plane for data center networks, and an end-host transport protocol that is explicitly aware of priority queues and employs a guided rate control mechanism.
- We conduct a comprehensive evaluation of PASE, which includes macro-benchmarks that compare PASE's performance against multiple existing transport protocols, and micro-benchmarks that focus on the internal working of the system.

PASE shows the promise of combining existing transport strategies in a single transport framework. We view it as a first step towards a more holistic approach for building the next generation data center transport protocols.

# 2. TRANSPORT STRATEGIES

To achieve high performance,[1] existing data center transports use one of the three following transport strategies: (a) *Self-Adjusting Endpoints*, (b) *Arbitration*, or (c) *In-network Prioritization*. We first describe these strategies and discuss their limitations when they are employed in isolation. We then discuss how these limitations can be addressed if these strategies are used together.

## 2.1 Transport Strategies in Isolation

Each transport strategy has its own advantages and disadvantages as shown in Table 1. We now describe the basic working of each strategy, discuss its advantages, and highlight their key limitations through simulation experiments.

**Self-Adjusting Endpoints:** Traditional transport protocols like TCP use this strategy. Under this transport strategy, endpoints *themselves* decide the amount of data to send based on network congestion. The state of network congestion is determined through a congestion signal that could be implicit (e.g., packet loss) or explicit (i.e., ECN). In case of congestion, the window size is reduced by the same factor for all flows, if fairness is the objective [11], or the factor could depend on other parameters (e.g., remaining flow size [22] or deadline [23]), if flow prioritization is the objective.

Protocols in this category are easy to deploy because they do not require any changes to the network infrastructure or depend on any entity except the endpoints. However, when considering flow prioritization, their performance is inferior to the state-of-the-art data center transport protocols (e.g., pFabric [12], PDQ [18]). One reason for their poor performance is that they do not provide strict priority scheduling – even low priority flows, which should be paused, continue to send at least one packet per RTT. This hurts performance, especially at high loads when multiple flows are active.

To illustrate this behavior, we consider two protocols that follow the self-adjusting endpoint strategy: DCTCP [11] and D$^2$TCP [23] (a deadline-aware version of DCTCP), and compare their performance with pFabric [12], the state-of-the-art data center transport with the best reported performance. We replicate a deadline oriented scenario in ns2[2]. Figure 1 shows the fraction of deadlines met (or *application throughput*) as a function of load for the three schemes. While at low loads, D$^2$TCP is able to meet deadlines (i.e., achieve prioritization), at higher loads its performance approaches its fair-sharing counterpart, DCTCP. Moreover, both these

[1]In the context of data center transports, *high performance* usually refers to minimizing completion times, maximizing throughput, or reducing the deadline miss rate [11, 18].

[2]This corresponds to Experiment 4.1.3 in the D$^2$TCP paper [23] – it represents an intra-rack scenario, where the source and destination of each flow is picked randomly and the flow sizes are uniformly distributed between [100 KB, 500 KB] in the presence of two background long flows. The deadlines are uniformly distributed from 5 ms-25 ms.

| Transport Strategy | Pros | Cons | Examples |
|---|---|---|---|
| Self-Adjusting Endpoints | Ease of deployment | Lack of support for strict priority scheduling | DCTCP [11], $D^2$TCP [23], $L^2$DCT [22] |
| Arbitration | 1) Supports strict priority scheduling<br>2) Fast convergence | 1) High flow switching overhead<br>2) Hard to compute precise rates | $D^3$ [24], PDQ [18]. |
| In-network Prioritization | 1) Work conservation<br>2) Low flow switching overhead | 1) Limited number of priority queues in existing switches<br>2) Switch-local decisions | pFabric [12] |

**Table 1: Comparison of different transport strategies.**

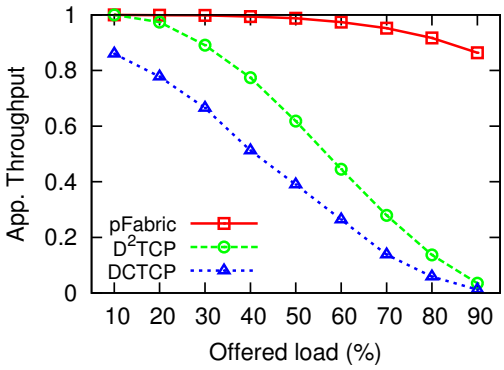



**Figure 1: Comparison of two self-adjusting endpoint based protocols ($D^2$TCP and DCTCP) with the state-of-the-art data center transport (pFabric).**

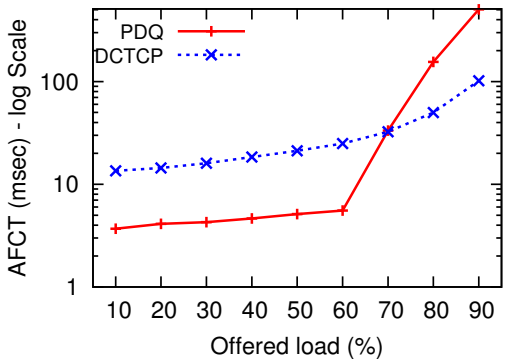



**Figure 2: Comparison of PDQ (an arbitration based approach) with DCTCP. At high loads, PDQ's high flow switching overhead leads to poor performance.**

protocols perform much worse than pFabric at high loads, highlighting their limitations in achieving priority scheduling across a wide range of network loads.

**Arbitration:** Explicit rate protocols, like PDQ [18], use arbitration as their underlying transport strategy. Instead of endpoints making decisions on their own, arbitration based approaches [18, 24] require the switches to make the scheduling decision, keeping in view all network flows and their individual priorities (e.g., deadline, flow size). The scheduling decision is communicated as a *rate* at which flows should send data. The rate could be zero, if the flow needs to be paused because of its low priority, or it could be the full link capacity, if the flow has the highest priority. While a centralized problem in general, prior work [18, 24] shows that arbitration can be done in a decentralized fashion – each switch along the path of a flow adds its rate to the packet header and the minimum rate is picked by the sender for transmitting data.

The explicit nature of arbitration based approaches ensures that flows achieve their desired rate quickly (typically in one RTT). Moreover, the ability to pause and unpause flows enables strict priority scheduling of flows: the highest priority flow gets the full link capacity (if it can saturate the link) while other flows are paused. However, this explicit rate assignment comes with its own set of problems. For example, calculating accurate rates for flows is challenging as flows could be bottlenecked at non-network resources (e.g., source application, receiver). Another important issue is the *flow switching overhead*, which refers to the overhead of pausing and unpausing flows. This overhead is typically ∼1-2 RTTs, which can be significant in scenarios involving short flows (when flows last for a small duration) and at high loads (when flows need to be frequently preempted).

We illustrate the impact of flow switching overhead in a practical scenario through a simulation experiment. We consider PDQ [18][3], which is considered the best performing arbitration based scheme, and compare its performance with DCTCP [11]. The scenario is a repeat of the previous intra-rack, all-to-all experiment, except the metric here is AFCT. Figure 2 shows the AFCT as a function of network load. At low loads, PDQ outperforms DCTCP because of its fast convergence to the desired rate. However, at high loads, the flow switching overhead becomes significant as more flows contend with each other, thereby requiring more preemptions in the network. As a result, PDQ's performance degrades and the completion time becomes even higher than that of DCTCP.

**In-network Prioritization:** In transport protocols that use in-network prioritization (e.g., pFabric [12]), packets carry flow priorities, such as the flow deadline or size, and switches use this priority to decide which packet to schedule or drop (in case of congestion). This behavior ensures two desirable properties: *work conservation*, a lower priority packet is scheduled if there is no packet of higher priority, and *preemption*, when a higher priority packet arrives, it gets precedence over a lower priority packet.

The well-known downside to in-network prioritization is the limited number of priority queues available in switches – typically ∼4-10 [24] (see Table 2). For most practical scenarios, this number is much smaller than the number of unique flow priorities in the system. Proposals that support a large number of *priority levels* require changing the network fabric [12], which makes them hard to deploy.

Another shortcoming of this strategy is that switches make local decisions about prioritization which can lead to sub-optimal performance in multi-link scenarios. This is shown in Figure 3 through a simple toy example involv-

[3]Based on the simulator code obtained from the PDQ authors. It supports all the optimizations that reduce the flow switching overhead.

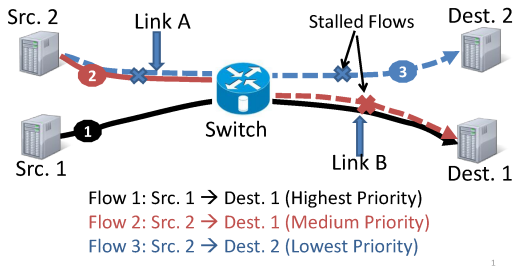



**Figure 3: Toy example illustrating problem with pFabric.**

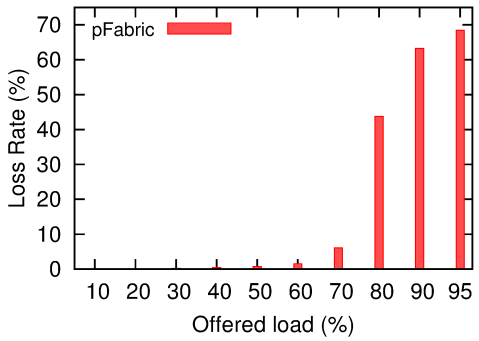



**Figure 4: Loss rate for pFabric with varying load. Local prioritization leads to losses at high loads.**

ing three flows. Flow 1 has the highest priority; Flow 2 has medium priority and flow 3 has the lowest priority. Flows 1 and 2 share link B, so only flow 1 can progress while flow 2 should wait. A protocol like pFabric continues to send packets of flow 2 on link A even though these packets are later dropped at link B. These unnecessary transmissions stall flow 3, which could have run in parallel with flow 1 as both flows do not share any link.

The above toy example highlights a common use case present in all-to-all traffic patterns (e.g., MapReduce [16], Search) where a node typically has data to send to many other nodes. To quantify this problem under such practical settings, we simulate the interaction between workers and aggregators within a single rack of a search application. Each worker-aggregator flow is uniformly distributed between [2-198] KB. We focus on the loss rate of pFabric when the network load is increased. As shown in Figure 4, loss rate shoots up as the load on network links is increased. For a load of 80%, more than 40% packets are dropped. These lost packets translate into loss of throughput as we could have used these transmissions for packets belonging to other flows. In Section 4, we show how this high loss rate results in poor FCT for pFabric.

## 2.2 Transport Strategies in Unison

We now discuss how *combining* these transport strategies offers a simple solution to the problems identified earlier.

**In-network Prioritization complementing arbitration:** The high flow switching overhead of arbitration-only approaches can be avoided with the help of in-network prioritization. As today's arbitration-only approaches, like PDQ [18], assume no prioritization within the network, they have to achieve priority scheduling by communicating explicit rates to end-hosts, which takes time, and thus results in a high flow switching overhead. If we have in-network prioritization, the arbitrator can just assign *relative* priorities to the flows (e.g., high priority flow vs low priority flow), leaving it up to the switches to *enforce* this relative priority through a suitable scheduling and dropping mechanism. The current in-network prioritization mechanisms (e.g., priority queues) provide seamless switching between flows of different priorities, so there is no flow switching overhead and link utilization remains high during this period.

A simple example illustrates this benefit. Assume we have two flows – $F_1$ (higher priority) and $F_2$ (lower priority). With arbitration-only approaches, $F_1$ is initially assigned the entire link capacity while $F_2$ is paused during this time. When $F_1$ finishes, we have to explicitly signal $F_2$ to unpause. With in-network prioritization, we can just assign these flows to different priority classes – $F_1$ is mapped to the high priority class while $F_2$ is mapped to the low priority class. The switch ensures that as soon as there are no more packets of $F_1$ in the high priority queue, it starts scheduling packets of $F_2$ from the lower priority queue.

**Arbitration aiding In-network Prioritization:** The small number of priority queues cause performance degradation when multiple flows get mapped to the high priority queue [12]. This results in multiplexing of these flows instead of strict priority scheduling (i.e., one flow at a time). This problem can be avoided with the help of arbitration. Instead of statically mapping flows to queues, an arbitrator can do a *dynamic* mapping. So a flow's priority queue keeps on changing during it's lifetime. A flow whose "turn" is far away is mapped to lower priority queue. As a flow's turn is about to come, it moves up to a higher priority queue.

We explain this idea through a simple two queue example. Suppose queue A ($Q_A$) is the high priority queue and queue B ($Q_B$) is the lower priority queue. We have four flows to schedule ($F_1$, $F_2$, $F_3$, and $F_4$, with $F_1$ having the highest priority and $F_4$, the lowest) — as the number of flows is more than the number of queues, any static mapping of flows to queues will result in sub-optimal performance. With the help of arbitration, we can initially map $F_1$ to $Q_A$ and the other three flows to $Q_B$. When $F_1$ finishes, we can change the mapping of $F_2$ from $Q_B$ to $Q_A$ while flows $F_3$ and $F_4$ are still mapped to $Q_B$. A similar process is applied when $F_2$ (and later on, $F_3$) finishes. In short, the highest priority queue is used for the active, high priority flow while the lower priority queue is used primarily to keep link utilization high (i.e., work-conservation). The example shows how arbitration can help leverage the limited number of priority queues without compromising on performance.

**Arbitration helping Self-Adjusting Endpoints :** With arbitration-only approaches, calculating precise flow rates can be hard because the arbitrator may not have accurate information about all the possible bottlenecks in the system [18, 24]. Thus, we can end up underestimating or overestimating the available capacity. Unfortunately, in arbitration-only approaches, endpoints – which typically have a better idea of path conditions – are dumb: they always transmit at the rate assigned by the arbitrator, so even if they are in a position to detect congestion or spare capacity in the network, they cannot respond.

The self-adjusting endpoint strategy naturally addresses this problem as it constantly probes the network: if there is any spare capacity, it will increase its rate, and if there is congestion, it will back off. For example, suppose there

| Switch | Vendor | Num. Queues | ECN |
|---|---|---|---|
| BCM56820 [2] | Broadcom | 10 | Yes |
| G8264 [4] | IBM | 8 | Yes |
| 7050S [1] | Arista | 7 | Yes |
| EX3300 [5] | Juniper | 5 | No |
| S4810 [3] | Dell | 3 | Yes |

**Table 2: Priority Queues and ECN support in popular commodity top-of-rack switches. The numbers are per interface.**

are two flows in the system with different priorities. The higher priority flow is assigned the full link capacity but it is unable to use it. The lower priority flow will remain paused if we do not use self-adjusting endpoints. However, if the endpoint uses a self-adjusting policy, it will detect spare capacity and increase its rate until the link is saturated. Note that arbitration also helps the self-adjusting endpoint strategy: instead of just blindly probing the network for its due share, a flow can use information from the arbitrator to "bootstrap" the self-adjusting behavior.

# 3. PASE

PASE is a transport framework that synthesizes the three transport strategies, namely in-network **P**rioritization, **A**rbitration, and **S**elf-adjusting **E**ndpoints. The underlying design principle behind PASE is that each transport strategy should focus on what it is best at doing, such as:

- Arbitrators should do inter-flow prioritization at coarse time-scales. They should not be responsible for computing precise rates or for doing fine-grained prioritization.
- Endpoints should probe for any spare link capacity on their own, without involving any other entity. Further, given their lack of global information, they should not try to do inter-flow prioritization (protocols that do this have poor performance, as shown in Section 2).
- In-network prioritization mechanism should focus on per-packet prioritization at short, sub-RTT timescales. The goal should be to obey the decisions made by the other two strategies while keeping the data plane simple and efficient.

Given the above roles for each strategy, the high-level working of PASE is as follows. Every source periodically contacts the arbitrator to get its *priority queue* and *reference rate*. The arbitration decision is based on the flows currently in the system and their priorities (e.g., deadline, flow-size). As the name suggests, the reference rate is not binding on the sources, so depending on the path conditions, the sources may end up sending at higher or lower than this rate (i.e., self-adjusting endpoints). A key benefit of PASE is that we do not require any changes to the data plane: switches on the path use existing priority queues to schedule packets and employ explicit congestion notification (ECN) to signal congestion. As shown in Table 2, most modern switches support these two features.

To achieve high performance while being deployment friendly, PASE incorporates two key components: a control plane arbitration mechanism and an end-host transport protocol. While existing arbitration mechanisms operate in the data plane (and hence require changes to the network fabric), we implement a separate control plane for performing arbitration in a scalable manner. To this end, we introduce optimizations that leverage the typical tree structure of data center topologies to reduce the overhead of arbitration. Finally, PASE's transport protocol has an explicit notion of reference rate and priority queues, which leads to new rate control and loss recovery mechanisms.

In the following sections, we describe the control plane and the end-host transport protocol of PASE, followed by the details of its implementation.

## 3.1 Arbitration Control Plane

While a centralized arbitrator is an attractive option for multiple reasons, making it work in scenarios involving short flows is still an open problem [10, 15]. Prior work [18] shows that the problem of flow arbitration can indeed be solved in a distributed fashion: each switch along the path of a flow independently makes the arbitration decision and returns the allocated rate for its own link, and the source can pick the minimum rate. While prior work implements arbitration as part of the data plane, PASE supports this as part of the control plane because experiences with prior protocols (e.g., XCP [20]) show that even small changes to the data plane are hard to deploy.

We first describe the basic arbitration algorithm used in PASE and the key sources of overhead that limit the scalability of the arbitration process. We then present two optimizations that reduce the arbitration overhead by exploiting the characteristics of typical data center topologies.

**Algorithm 1** Arbitration Algorithm

**Input:** $< FlowSize, FlowID, demand(optional) >$
**Output:** $< PrioQue, R_{ref} >$
Arbitrator locally maintains a sorted list of flows
**Step#1:** Sort/update flow entry based on the $FlowSize$
**Step#2:** Compute $PrioQue$ and $R_{ref}$ of the flow.
//Link capacity is $C$, and AggregateDemandHigher ($ADH$) is the sum of demands of flows with priority higher than current flow.
1: Assign rate:
**if** $ADH < C$ **then**
$R_{ref} = min(demand, C - ADH)$;
**else**
$R_{ref} = baserate$;
**end if**
2: Assign $PrioQue$:
$PrioQue = \lceil ADH/C \rceil$;
**if** $PrioQue > LowestQueue$ **then**
$PrioQue = LowestQueue$;
**end if**
3: return $< PrioQue, R_{ref} >$;

### 3.1.1 Basic Arbitration Algorithm

For each link in the data center topology, there is an arbitrator that runs the arbitration algorithm and makes decisions for all flows that traverse the particular link. The arbitrator can be implemented on any server in the data center or on dedicated controllers that are co-located with the switches. The arbitration algorithm works at the granularity of a flow, where a flow could be a single RPC in a typical client-server interaction, or a long running TCP-like connection between two machines.

The interaction between a flow's source and the arbitrator(s) is captured in Algorithm 1. The source provides the arbitrator(s) with two pieces of information: i) the flow size ($FlowSize$), which is used as the criterion for scheduling (i.e., shortest flow first). To support other scheduling techniques, the $FlowSize$ can be replaced by deadline [24] or task-id for task-aware scheduling [17]. ii) demand, this represents the maximum rate at which the source can send data. For long flows that can saturate the link, this is equal to the NIC rate, while for short flows that do not have enough data to saturate the link, this is set to a lower value. Demand and flow size are inputs to the arbitration algorithm whereas the output is the reference rate ($R_{ref}$) and a priority queue ($PrioQue$) of the flow.

To compute $R_{ref}$ and $PrioQue$, the arbitrator locally maintains a sorted list of flows based on their sizes. This list is updated based on the latest $FlowSize$ information of the current flow. The flow's priority queue and reference rate depend on the aggregate demand (AggregateDemandHigher - or ADH) of flows that have higher priority compared to the current flow. An ADH value less than the link capacity $C$ implies that there is some spare capacity on the link and the flow can be mapped to the top queue. Thus, if the flow's demand is less than the spare capacity, we set the reference rate equal to the demand. Otherwise, we set the reference rate equal to the spare capacity.

The other case is when the ADH exceeds link capacity. This happens when a link is already saturated by higher priority flows, so the current flow cannot make it to the top queue. In this case, the flow's reference rate is set to a base value, which is equal to one packet per RTT. This allows such low priority flows to make progress in case some capacity becomes available in the network, and to even increase their rate in the future based on self-adjusting behavior.

Finally, if the current flow cannot be mapped to the top queue, it is either mapped to the lowest queue (if ADH exceeds the aggregate capacity of intermediate queues) or is mapped to one of the intermediate queues. Thus, each intermediate queue accommodates flows with an aggregate demand of $C$ and the last queue accommodates all the remaining flows.

**Challenges.** While the above design provides a simple distributed control plane for arbitration, it has three sources of overhead that limit its scalability.

- Communication Latency. The communication latency between the source and the arbitrator depends on their physical distance within the data center. This delay matters the most during flow setup time as it can end up increasing the FCT, especially for short flows. To keep this delay small, arbitrators must be placed carefully, such that they are located as close to the sources as possible.
- Processing Overhead. The second challenge is the processing overhead of arbitration messages, which can potentially add non-negligible delay, especially under high load scenarios.
- Network Overhead. Due to a separate control plane, each arbitration message is potentially processed as a separate packet by the switches which consumes link capacity. We need to ensure that this overhead is kept low and that it does not cause network congestion for our primary traffic.

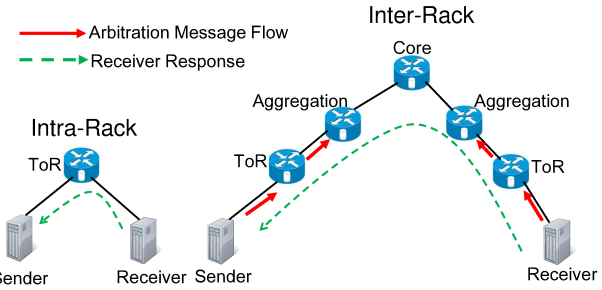



**Figure 5: The bottom-up approach employed by PASE for intra-rack and inter-rack scenarios.**

To reduce these overheads, we extend the basic PASE design by introducing a bottom-up approach to arbitration, which we describe next.

### 3.1.2 Bottom Up Arbitration

We exploit the typical tree structure of data center topologies. Sources obtain information about the $PrioQue$ and $R_{ref}$ in a bottom-up fashion, starting from the leaf nodes up to the core as shown in Figure 5. For this purpose, the end-to-end path between a source and destination is divided into two halves – one from the source up to the root and the other from the destination up to the root.

For each half, the respective leaf nodes (i.e., source and destination) initiate the arbitration. They start off with their link to the ToR switch and then move upwards. The arbitration request messages move up the arbitration hierarchy until it reaches the top level arbitrator. The responses move downwards in the opposite direction.

This bottom-up approach ensures that for intra-rack communication, arbitration is done solely at the endpoints, without involving any external arbitrator. Thus, in this scenario, flows incur no additional network latency for arbitration. This is particularly useful as many data center applications have communication patterns that have an explicit notion of rack affinity [11, 13].

For the inter-rack scenario, the bottom up approach facilitates two other optimizations, *early pruning* and *delegation*, to reduce the arbitration overhead and latency. Both early pruning and delegation exploit a trade-off between low overhead and high accuracy of arbitration. As our evaluation shows, by giving away some accuracy, they can significantly decrease the arbitration overhead.

**Early Pruning.** The network and processing overheads can be reduced by limiting the number of flows that contact the arbitrators for $PrioQue$ and $R_{ref}$ information. In early pruning, only flows that are mapped to the highest priority queue move upwards for arbitration. Thus, in Algorithm 1 a lower level arbitrator only sends the arbitration message to its parent if the flow is mapped to the top queue(s). This results in lower priority flows being pruned at lower levels, as soon as they are mapped to lower priority queues (see Figure 6). The intuition behind this is that a flow mapped to a lower priority queue on one of its links will never make it to the highest priority queue irrespective of the arbitration decision on other links. This is because a flow always uses the lowest of the priority queues assigned by all the arbitrators (i.e., bottleneck in the path). Thus, we should avoid the overhead of making arbitration decisions for the flows mapped to lower priority queues.

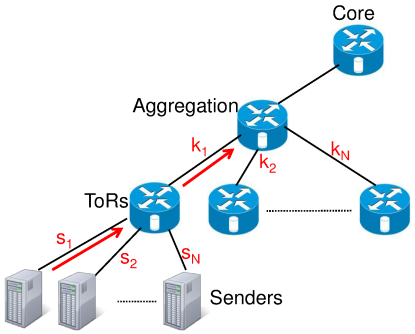



**Figure 6: The *early pruning* optimization used by PASE for reducing the arbitration overhead. Note that $s_i$ and $k_i$ represent flows that are mapped to the highest priority queue at the senders and the ToR arbitrators, respectively.**

There are two key benefits of early pruning. First, it reduces the network overhead as arbitration messages for only high priority flows propagate upwards. Second, it reduces the processing load on higher level arbitrators. In both cases, the reduction in overhead can be significant, especially in the more challenging heavy load scenarios, where such overhead can hurt system performance.

In general, early pruning makes the overhead independent of the total number of flows in the system. Instead, with early pruning, the overhead depends on the number of children that a higher level arbitrator may have because each child arbitrator only sends a limited number of flows (the top ones) upwards. Due to limited port density of modern switches (typically less than 100), the number of children, and hence the overhead, is quite small. This leads to significant reduction in the overhead compared to the non-pruning case where the overhead is proportional to the total number of flows traversing a link, which can be in thousands (or more) for typical data center settings [11, 13]. However, the above overhead reduction comes at the cost of less precise arbitration. As we only send the top flows to the higher level arbitrators, flows that are pruned do not get the complete and accurate arbitration information. Our evaluation shows that sending flows belonging to the top two queues upwards (rather than just the top queue), provides the right balance: there is little performance degradation while reduction in overhead is still significant.

**Delegation.** Delegation is specially designed to reduce the arbitration latency. While early pruning can significantly reduce the arbitration processing overhead as well as the network overhead, it does not reduce the latency involved in the arbitration decision because top flows still need to go all the way up to the top arbitrator.

In delegation, a higher level link (i.e., closer to the core) is divided into smaller "virtual" links of lower capacity – each virtual link is delegated to one of the child arbitrators who then becomes responsible for all arbitration decisions related to the virtual link. Thus, it can make a local decision about a higher level link without going to the corresponding arbitrator. On each virtual link, we run the same arbitration algorithm i.e., Algorithm 1 as we do for normal links.

As a simple example, the aggregation-core link of capacity $C$ shown in Figure 7 can be divided into $N$ virtual links of capacity $a_iC$ each (where $a_i$ is the fraction of capacity allocated to virtual link $i$) and then delegated to one of the child

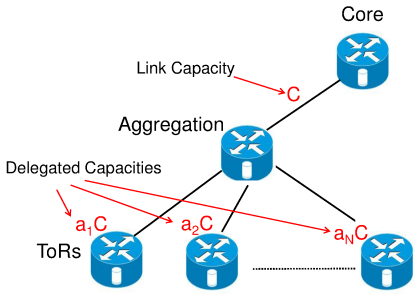



**Figure 7: The *delegation* optimization used by PASE for reducing the setup latency and control overhead.**

arbitrators. The capacity of each virtual link is updated periodically, reflecting the *PrioQue* of flows received by each child arbitrator. For example, one child that is consistently observing flows of higher priorities can get a virtual link of higher capacity.

Delegation provides two benefits. First, it reduces the flow setup delay because arbitration decisions for higher level links are made at lower level arbitrators, which are likely to be located close to the sources (e.g., within the rack). Second, it reduces the control traffic destined towards higher level arbitrators. Note that this happens because only *aggregate* information about flows is sent by the child arbitrators to their parents for determining the new share of virtual link capacities.

The impact on processing load has both positive and negative dimensions. While it reduces the processing load on higher level arbitrators, it ends up increasing the load on lower level arbitrators as they need to do arbitration for their parents' virtual links too. However, we believe this may be acceptable as lower level arbitrators typically deal with fewer flows compared to top level arbitrators.

Like early pruning, delegation also involves the overhead-accuracy trade-off. The capacity assigned to a specific virtual link may not be accurate which may lead to performance degradation. For example, we may have assigned a lower virtual capacity to a child who may suddenly start receiving higher priority flows. These flows would need to wait for an update to the virtual link capacity before they can get their due share. On the other hand, there could be cases where the virtual capacity may remain unused if the child does not have enough flows to use this capacity. This is especially true in scenarios where a link is delegated all the way to the end-host and the end-host may have a bursty flow arrival pattern.

Given the above trade-off, PASE only delegates the Aggregation-Core link capacity to its children (TOR-Aggregation arbitrators). These child arbitrators should be typically located within the rack of the source/destination (or co-located with the TOR switch). Thus, for any inter-rack communication within a typical three level tree topology, the source and destination only need to contact their TOR-aggregation arbitrator, who can do the arbitration all the way up to the root. In fact, flows need not wait for the feedback from the other half i.e., destination-root. Thus, in PASE, a flow starts as soon as it receives arbitration information from the child arbitrator. This approach is reasonable in scenarios where both halves of the tree are likely to have similar traffic patterns. If that is not true then PASE's self-adjusting behavior ensures that flows adjust accordingly.

## 3.2 End-host Transport

PASE's end-host transport builds on top of existing transports that use the self-adjusting endpoints strategy (e.g., DCTCP). Compared to existing protocols, the PASE transport has to deal with two new additional pieces: (a) a priority queue ($PrioQue$) on which a flow is mapped and (b) a reference sending rate ($R_{ref}$). This impacts two aspects of the transport protocol: rate control and loss recovery. Rate control in PASE is more guided and is closely tied to the $R_{ref}$ and the $PrioQue$ of a flow. Similarly, the transport requires a new loss recovery mechanism because flows mapped to lower priority queues may experience spurious timeouts as they have to wait for a long time before they get an opportunity to send a packet. We now elaborate on these two aspects.

**Rate Control:** A PASE source uses the $R_{ref}$ and the $PrioQue$ assigned by the arbitrators to guide its transmission rate. The rate control uses congestion window ($cwnd$) adjustment, based on the $R_{ref}$ and the flow RTT, to achieve the average reference rate at RTT timescales.

Algorithm 2 describes the rate control mechanism in PASE. For the flows mapped to the top queue, the congestion window is set to $R_{ref} \times RTT$ in order to reflect the reference rate assigned by the arbitrator. For all other flows, the congestion window is set to one packet. Note that for flows belonging to the top queue, the reference rate $R_{ref}$ is generally equal to the access link capacity unless flows have smaller sizes.

For the flows mapped to lower priority queues (except the bottom queue), the subsequent increase or decrease in congestion window $cwnd$ is based on the well-studied control laws of DCTCP [11]. In particular, when an unmarked ACK is received, the window size is increased as

$$cwnd = cwnd + 1/cwnd. \quad (1)$$

When a marked ACK (i.e., with ECN-Echo flag set) is received, the window size is reduced as

$$cwnd = cwnd \times (1 - \alpha/2) \quad (2)$$

where $\alpha$ is the weighted average of the fraction of marked packets. This self-adjusting behavior for higher priority queues is important for ensuring high fabric utilization at all times because the $R_{ref}$ may not be accurate and there may be spare capacity or congestion along the path.

For the flows mapped to the bottom queue, the window size always remains one. This is because under high loads all flows that cannot make it to the top queues are mapped to the bottom queue, so the load on the bottom queue can be usually high.

**Loss Recovery:** For flows belonging to the top queue, we use existing loss recovery mechanisms (i.e., timeout based retransmissions). However, flows that get mapped to the lower priority queues can timeout for two reasons: (a) their packet(s) could not be transmitted because higher priority queues kept the link fully utilized and (b) a packet was lost. In case of scenario (a), a sender should avoid sending any new packets into the network as it increases the buffer occupancy and the likelihood of packet losses especially at high network loads. In case of scenario (b), a sender should retransmit the lost packet as early as it is possible so that the flows can make use of any available capacity without under-utilizing the network. However, differentiating between these two scenarios is a challenging task without incurring any additional overhead.

**Algorithm 2** Rate Control

**Input:** $< PrioQue, R_{ref} >$
**Output:** $< cwnd >$ // congestion window
// Priority queues $q_1$, $q_2$,.., $q_k$ where $q_1$ is the highest priority queue and $q_2$, $q_3$, and $q_{k-1}$ are intermediate queues
// if an ACK with the ECN-Echo flag set is received
**if** $ACK_{marked} == 1$ **then**
$cwnd = cwnd \times (1 - \alpha/2)$; // Use DCTCP decrease law
**else**
**if** $PrioQue == q_1$ **then**
$cwnd = R_{ref} \times RTT$;
$isInterQueue = 0$; // not an intermediate queue
**else if** $PrioQue \in \{q_2, q_3, .., q_{k-1}\}$ **then**
// if already mapped to an intermediate queue
**if** $isInterQueue == 1$ **then**
$cwnd = 1 + 1/cwnd$; // Use DCTCP increase law
**else**
$isInterQueue = 1$; $cwnd = 1$;
**end if**
**else if** $PrioQue == q_k$ **then**
$cwnd = 1$; $isInterQueue = 0$;
**end if**
**end if**

We use small probe packets instead of retransmitting the entire packet whenever a flow, mapped to one of the lower priority queues, times out. If we receive an ACK for the probe packet (but not the original packet), it is a sign that the data packet was lost, so we retransmit the original data packet. If the probe packet is also delayed (no ACK received), we increase our timeout value (and resend another probe) just like we do this for successive losses for data packets. An alternative approach is to set suitable timeout values: smaller values for flows belonging to the top queue and larger values for the other flows.

Finally, a related issue is that of *packet reordering*. When a flow from a low priority queue gets mapped to a higher priority queue, packet re-ordering may occur as earlier packets may be buffered at the former queue. This can lead to unnecessary backoffs which degrades throughput. To address this, we ensure that when a flow moves to a higher priority queue, we wait for the ACKs of already sent packets to be received before sending packets with the updated priority.

## 3.3 Implementation

We implement PASE on a small-scale Linux testbed as well as in the ns2 simulator. Our ns2 implementation supports all the optimizations discussed in Section 3 whereas our testbed implementation only supports the basic arbitration algorithm (described in Section 3.1.1). For the testbed evaluation, we implement the transport protocol and the arbitration logic as Linux kernel modules. The Linux transport module, which is built on top of DCTCP, communicates with the arbitration module to obtain the packet priority and the reference rate. It then adds the PASE header on outgoing packets. For the desired switch support, we use the PRIO [7] queue and CBQ (class based queueing) implementation, on top of the RED queue implementation in Linux and ns2. We use eight priority queues/classes and

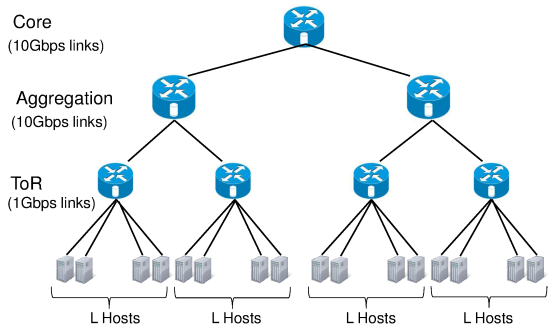



**Figure 8: Baseline topology used in simulations. Note that $L = 40$ hosts.**

classify packets based on the ToS field in the IP header. Out of these eight queues, a separate, strictly lower priority queue is maintained for background traffic. For the RED queue, we set the low and high thresholds of RED queue to $K$ and perform marking based on the instantaneous rather than the average queue length as done in DCTCP [11].

# 4. EVALUATION

In this section we evaluate the performance of PASE using the ns2 simulator [6] as well as through small-scale testbed experiments. First, we conduct macro-benchmark experiments in ns2 to compare PASE's performance against existing data center transports. We compare PASE against both deployment-friendly protocols including DCTCP [11], D$^2$TCP [23], and L$^2$DCT [22] (§4.2.1) as well as the best performing transport, namely, pFabric [12] (§4.2.2). Second, we micro-benchmark the internal working of PASE (e.g., benefits of arbitration optimizations, use of reference rate, etc) using simulations (§4.3). Finally, we report evaluation results for our testbed experiments (§4.4).

## 4.1 Simulation Settings

We now describe our simulation settings including the data center topology, traffic workloads, performance metrics, and the protocols compared.

**Data center Topology:** We use a 3-tier topology for our evaluation comprising layers of ToR (Top-of-Rack) switches, aggregation switches, and a core switch as shown in Figure 8. This is a commonly used topology in data centers [9, 23, 24]. The topology interconnects 160 hosts through 4 ToR switches that are connected to 2 aggregation switches, which in turn are interconnected via a core switch. Each host-ToR link has a capacity of 1 Gbps whereas all other links are of 10 Gbps. This results in an oversubscription ratio of 4:1 for the uplink from the ToR switches. In all the experiments, the arbitrators are co-located with their respective switches. The end-to-end round-trip propagation delay (in the absence of queueing) between hosts via the core switch is 300$\mu$s.

**Traffic Workloads:** We consider traffic workloads that are derived from patterns observed in production data centers. Flows arrive according to a Poisson process and flow sizes are drawn from the interval [2 KB, 198 KB] using a uniform distribution, as done in prior studies [18, 24]. This represents query traffic and latency sensitive short messages in data center networks. In addition to these flows, we generate two long-lived flows in the background, which represents the 75$^{th}$ percentile of multiplexing in data centers [11]. Note that we always use these settings unless specified otherwise.

| Scheme | Parameters |
|---|---|
| DCTCP | qSize = 225 pkts |
| D$^2$TCP | markingThresh = 65 |
| L$^2$DCT | minRTO = 10 ms |
| pFabric | qSize = 76 pkts (= 2×BDP)<br>initCwnd = 38 pkts (= BDP)<br>minRTO = 1 ms (~3.3×RTT) |
| PASE | qSize = 500 pkts<br>minRTO (flows in top queue) = 10 ms<br>minRTO (flows in other queues) = 200 ms<br>numQue = 8 |

**Table 3: Default simulation parameter settings.**

We consider two kinds of flows: *deadline-constrained flows* and *deadline-unconstrained flows*. They cover typical application requirements in today's data centers [23].

**Protocols Compared:** We compare PASE with several data center transports including DCTCP [11], D$^2$TCP [23], L$^2$DCT [22], and pFabric [12]. We implemented DCTCP, D$^2$TCP and L$^2$DCT in ns2 and use the source code of pFabric provided by the authors to evaluate their scheme. The parameters of these protocols are set according to the recommendations provided by the authors, or reflect the best settings, which we determined experimentally (see Table 3).

**Performance Metrics:** For traffic without any deadlines, we use the FCT as a metric. We consider the AFCT as well as the 99$^{th}$ percentile FCT for small flows. For deadline-constrained traffic, we use application throughput as our metric which is defined as the fraction of flows that meet their deadlines. We use the control messages per second to quantify the arbitration overhead.

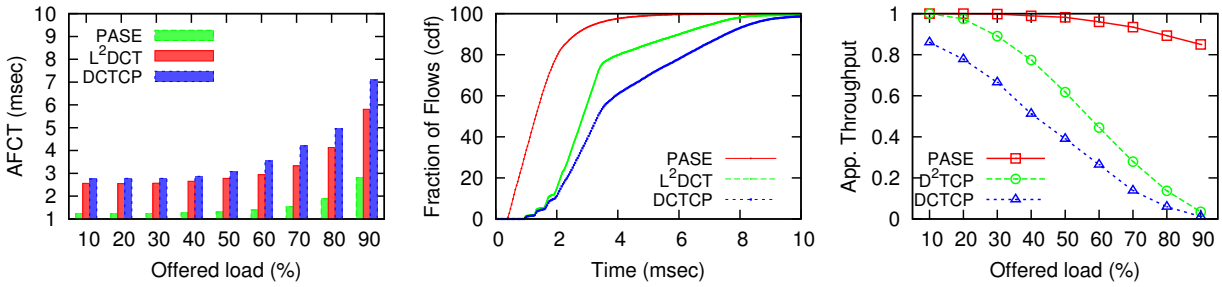
## 4.2 Macro-benchmarks

### 4.2.1 Comparison with Deployment Friendly Schemes

PASE is a deployment friendly transport that does not require any changes to the network fabric. Therefore, we now compare PASE's performance with deployment friendly data center transports, namely, DCTCP and L$^2$DCT. DCTCP [11] is a *fair sharing* protocol that uses ECN marks to infer the degree of congestion and employs adaptive backoff factors to maintain small queues. The goal of L$^2$DCT [22] is to minimize FCT – it builds on DCTCP by *prioritizing* short flows over long flows through the use of adaptive control laws that depend on the size of the flows.

**Deadline-unconstrained flows:** We consider an inter-rack communication scenario (termed as *left-right*) where 80 hosts in the left subtree of the core switch generate traffic towards hosts in the right subtree. This is a common scenario in user-facing web services where the front-end servers and the back-end storage reside in separate racks [25]. The generated traffic comprises flows with sizes drawn from the interval [2 KB, 198 KB] using a uniform distribution. In addition, we generate two long-lived flows in the background.

Figure 9(a) shows the improvement in AFCT as a function of network load. Observe that PASE outperforms L$^2$DCT and DCTCP by at least 50% and 70%, respectively across a wide range of loads. At low loads, PASE performs better primarily because of its quick convergence to the correct rate for each flow. At higher loads, PASE ensures that shorter flows are *strictly* prioritized over long flows, whereas with L$^2$DCT, all flows, irrespective of their priorities, continue to send at least one packet into the network. This lack




(a) AFCT (b) CDF of FCTs (c) Application Throughput

**Figure 9: (a) Comparison of PASE with L$^2$DCT and DCTCP in terms of AFCTs under the (*left-right*) inter-rack scenario, (b) shows the CDF of FCTs at 70% load in case of (a), and (c) Deadline-constrained flows: Comparison of application throughput for PASE with D$^2$TCP and DCTCP under the intra-rack scenario.**

of support for strict priority scheduling in L$^2$DCT leads to larger FCTs for short flows. DCTCP does not prioritize short flows over long flows. Thus, it results in worst FCTs across all protocols.

Figure 9(b) shows the CDF of FCTs at 70% load. Observe that PASE results in better FCTs for almost all flows compared to L$^2$DCT and DCTCP.

***Deadline-constrained flows:*** We now consider latency-sensitive flows that have specific deadlines associated with them. Thus, we compare PASE's performance with D$^2$TCP, a deadline-aware transport protocol. We replicate the D$^2$TCP experiment from [23] (also described earlier in §2) which considers an intra-rack scenario with 20 machines and generate short query traffic with flow sizes drawn from the interval [100 KB, 500 KB] using a uniform distribution. Figure 9(c) shows the application throughput as a function of network load. PASE significantly outperforms D$^2$TCP and DCTCP, especially at high loads because of the large number of active flows in the network. Since each D$^2$TCP and DCTCP flow sends at least one packet per RTT, these flows consume significant network capacity which makes it difficult for a large fraction of flows to meet their respective deadlines. PASE, on the other hand, ensures that flows with the earliest deadlines are given the desired rates and are strictly prioritized inside the switches.

### 4.2.2 Comparison with Best Performing Scheme

We now compare the performance of PASE with pFabric, which achieves close to optimal performance in several scenarios but requires changes in switches. With pFabric, packets carry a priority number that is set independently by each flow. Based on this, pFabric switches perform priority-based scheduling and dropping. All flows start at the line rate and backoff only under persistent packet loss.

PASE performs better than pFabric in two important scenarios: (a) multi-link (single rack) scenarios with all-to-all traffic patterns and (b) at high loads (generally $>80\%$) whereas it achieves similar performance ($<6\%$ difference in AFCTs) compared to pFabric in the following two cases: (a) single bottleneck scenarios and (b) when the network load is typically less than 80%.

We first consider the *left-right* inter-rack scenario where the aggregation-core link becomes the bottleneck. Figure 10(a) shows the $99^{th}$ percentile FCT as a function of load. Observe that pFabric achieves smaller FCT for up to 50% load and PASE achieves comparable performance. However, at $\geq 60\%$ loads, PASE results in smaller FCT than pFabric. At 90% load, this improvement is more than 85%. This happens due to high and persistent loss rate with pFabric at high loads. Figure 10(b) shows the CDF of FCT at 70% load for the same scenario.

Next we consider an all-to-all intra-rack scenario, which is common in applications like web search where responses from several worker nodes within a rack are combined by an aggregator node before a final response is sent to the user. Moreover, any node within a rack can be an aggregator node for a user query and the aggregators are picked in a round robin fashion to achieve load balancing [8].

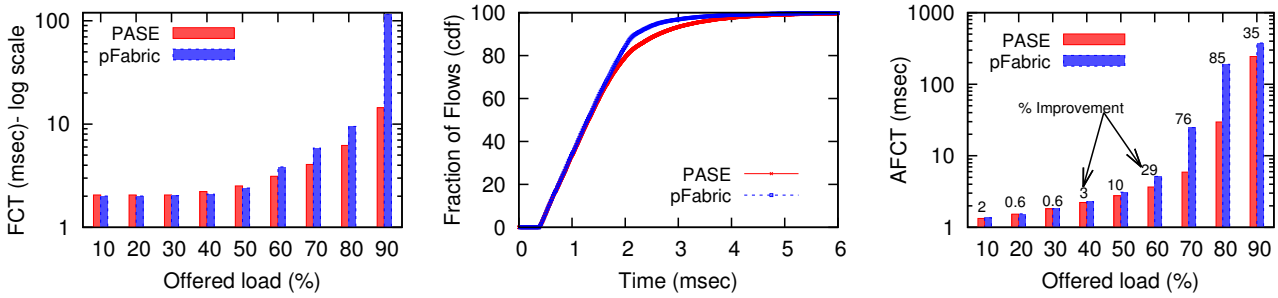
Figure 10(c) shows that PASE provides up to 85% improvement in AFCT over pFabric and results in lower AFCTs across *all* loads. This happens because with pFabric multiple flows sending at line rate can collide at a downstream ToR-host link. This causes a significant amount of network capacity to be wasted on the host-ToR links, which could have been used by other flows (as this is an all-to-all scenario). With PASE, flows do not incur any arbitration latency in the intra-rack scenario as new flows start sending traffic at line rate based on the information (priority and reference rate) from their local arbitrator. After one RTT, all flows obtain their global priorities which helps in avoiding any persistent loss of throughput in case the local and global priorities are different.

## 4.3 Micro-benchmarks

In this section, we evaluate the basic components of PASE with the help of several micro-benchmarks. Our results show that PASE optimizations significantly reduce the control traffic overhead and the number of messages that arbitrators need to process. In addition, we find that other features of PASE such as its rate control are important for achieving high performance.

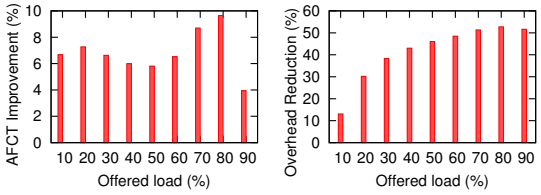
### 4.3.1 Arbitration Optimizations

PASE introduces early pruning and delegation for reducing the arbitration overhead of update messages. We now evaluate the overhead reduction brought about by these optimizations as well as study their impact on the performance of PASE. Figure 11(b) shows the overhead reduction that is achieved by PASE when all its optimizations are enabled. Observe that these optimizations provide up to 50% reduction in arbitration overhead especially at high loads. This happens because when these optimizations are enabled, higher-level arbitrators delegate some portion of the bandwidth to lower level arbitrators, which significantly reduces



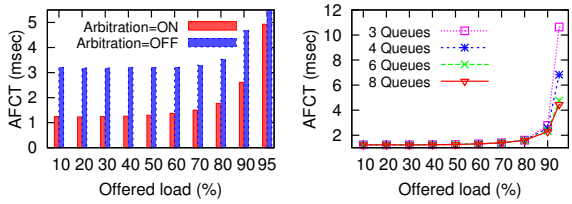

(a) *left-right* inter-rack (b) *left-right* inter-rack (c) all-to-all intra-rack

**Figure 10: Comparison of PASE with pFabric under (a) $99^{th}$ percentile FCT in the *left-right* inter-rack scenario, (b) CDF of FCTs under the *left-right* scenario at 70% load, and (c) all-to-all intra-rack scenario.**




(a) AFCT (b) Overhead

**Figure 11: AFCT improvement and overhead reduction as a function of load for PASE with its optimizations in the *left-right* scenario.**

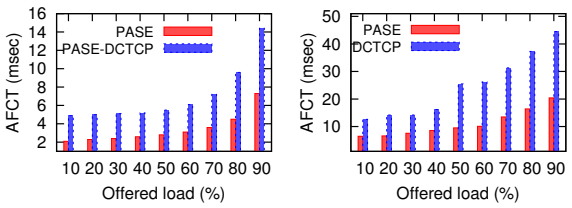



(a) AFCT (b) Priority Queues

**Figure 12: (a) Comparison of AFCT with and without end-to-end arbitration and (b) PASE with varying number of priority queues (*left-right* scenario).**

the control overhead on ToR-Aggregation links. In addition, updates of only those flows are propagated that map to the highest priority queues due to early pruning. Figure 11(a) shows that these optimizations also improve the AFCT by 4-10% across all loads. This happens because of delegation, which reduces the arbitration delays.

**Benefit of end-to-end arbitration:** PASE enables global prioritization among flows through its scalable end-to-end arbitration mechanism. This arbitration, however, requires additional update messages to be sent on the network. It is worth asking if most of the benefits of PASE are realizable through only local arbitration, which can be solely done by the endpoints. Thus, we now compare the performance of end-to-end arbitration and local arbitration in the *left-right* inter-rack scenario. Figure 12(a) shows that end-to-end arbitration leads to significant improvements (up to 60%) in AFCTs across a wide range of loads. This happens because local arbitration cannot account for scenarios where contention does not occur on the access links, thus leading to sub-optimal performance.




(a) PASE vs PASE-DCTCP (b) Testbed Evaluation

**Figure 13: (a) Comparison of PASE with PASE-DCTCP in the intra-rack scenario with 20 nodes and uniformly distributed flow sizes from [100 KB, 500 KB] and (b) Testbed Evaluation: Comparison of AFCT for PASE with DCTCP.**

### 4.3.2 *In-network Prioritization and Transport Micro-benchmarks*

**Impact of number of priority queues:** We now evaluate the impact of changing the number of priority queues in the switches. To test this scenario we repeat the *left-right* inter-rack scenario with different number of queues. Figure 12(b) shows that using 4 queues provide significant improvement in AFCT at loads $\geq 70\%$. However, increasing the number of queues beyond this provides only marginal improvement in AFCT. These results further reinforce the ability of PASE to achieve high performance with existing switches that support a limited number of priority queues.

**Reference Rate:** We now evaluate the benefit of using the reference rate information. We compare PASE with PASE-DCTCP, where *all* flows (including the ones mapped to the top queue as well as the lowest priority queue) behave as DCTCP sources and do not use the reference rate. However, these flows are mapped to different priority queues through the normal arbitration process. As shown in Figure 13(a), leveraging reference rate results in AFCTs of PASE to be 50% smaller than the AFCTs for PASE-DCTCP.

**Impact of RTO and Probing:** Flows that are mapped to the lower priority queues may experience large number of timeouts, which can affect performance. We implemented probing in which flows mapped to the lowest priority queue send a header-only probe packet every RTT rather than a full-sized packet. We found that using probing improves performance by ≈2.4% and ≈11% at 80% and 90% loads, respectively in the all-to-all intra-rack scenario. Note that unlike pFabric, PASE does not require small RTOs which forgoes the need to have high resolution timers.

## 4.4 Testbed Evaluation

We now present a subset of results from our testbed evaluation. Our testbed comprises of a single rack of 10 nodes (9 clients, one server), with 1 Gbps links, 250 $\mu$sec RTT and a queue size of 100 packets on each interface. We set the marking threshold $K$ to 20 packets and use 8 priority queues. We compare PASE's performance with the DCTCP implementation (provided by its authors). To emulate data center like settings, we generate flows sizes that are uniformly distributed between 100 KB and 500 KB, as done in [23]. We start 1000 short flows and vary the flow arrival rate to generate a load between 10% to 90%. In addition, we also generate a long lived background flow from one of the clients. We compare PASE with DCTCP and report the average of ten runs. Figure 13(b) shows the AFCT for both the schemes. Observe that PASE significantly outperforms DCTCP: it achieves ≈50%-60% smaller AFCTs compared to DCTCP. This also matches the results we observed in ns2 simulations.

# 5. RELATED WORK

We now briefly describe and contrast our work with the most relevant research works. We categorize prior works in terms of the underlying transport strategies they use.

**Self-Adjusting Endpoints:** Several data center transports use this strategy [11, 14, 22, 23]. DCTCP uses an adaptive congestion control mechanism based on ECN to maintain low queues. D$^2$TCP and L$^2$DCT add deadline-awareness and size-awareness to DCTCP, respectively. MCP [14] improves performance over D$^2$TCP by assigning more precise flow rates using ECN marks. These rates are based on the solution of a stochastic delay minimization problem. These protocols do not support flow preemption and in-network prioritization, which limits their performance.

**Arbitration:** PDQ [18] and D$^3$ [24] use network-wide arbitration but incur high flow switching overhead. PASE's bottom up approach to arbitration has similarities with EyeQ [19], which targets a different problem of providing bandwidth guarantees in multi-tenant cloud data centers. PASE's arbitration mechanism generalizes EyeQ's arbitration by dealing with scenarios where contention can happen at links other than the access links.

**In-network Prioritization:** pFabric [12] uses in-network prioritization by doing priority-based scheduling and dropping of packets. DeTail [25], a cross layer network stack, focuses on minimizing the *tail* latency but does not target the average FCT. In [21], authors propose virtual shapers to overcome the challenge of limited number of rate limiters. While both DeTail and virtual shapers use in-network prioritization, they do not deal with providing mechanisms for achieving network-wide arbitration.

In general, prior proposals target one specific transport strategy, PASE uses all these strategies in unison to overcome limitations of individual strategies and thus achieves high performance across a wide range of scenarios while being deployment friendly.

# 6. CONCLUSION

We proposed PASE, a transport framework that synthesizes existing transport strategies. PASE is deployment friendly as it does not require any changes to the network fabric and yet, its performance is comparable to, or better than the state-of-the-art protocols that require changes to network elements. The design of PASE includes a scalable arbitration control plane which is specifically tailored for typical data center topologies, and an end-host transport that explicitly uses priority queues and information from arbitrators. We believe that PASE sets out a new direction for data center transports, where advances in specific techniques (e.g., better in-network prioritization mechanisms or improved control laws) benefit everyone.

**Acknowledgements:** We thank our shepherd, Nandita Dukkipati, and the SIGCOMM reviewers for their feedback. We also thank Thomas Karagiannis and Zafar Ayyub Qazi for their useful comments.

# 7. REFERENCES

[1] Arista 7050s switch. http://www.aristanetworks.com/docs/Manuals/ConfigGuide.pdf.
[2] Broadcom bcm56820 switch. http://www.broadcom.com/collateral/pb/56820-PB00-R.pdf.
[3] Dell force10 s4810 switch. http://www.force10networks.com/CSPortal20/KnowledgeBase/DOCUMENTATION/CLIConfig/FTOS/Z9000_CLI_8.3.11.4_23-May-2012.pdf.
[4] Ibm rackswitch g8264 application guide. http://www.bladenetwork.net/userfiles/file/G8264_AG_6-8.pdf.
[5] Juniper ex3300 switch. http://www.juniper.net/us/en/products-services/switching/ex-series/ex3300/.
[6] The network simulator - ns-2. http://www.isi.edu/nsnam/ns/.
[7] Prio qdisc linux. http://linux.die.net/man/8/tc-prio, 2006.
[8] D. Abts and B. Felderman. A guided tour of data-center networking. *Commun. ACM*, 55(6):44–51, June 2012.
[9] M. Al-Fares, A. Loukissas, and A. Vahdat. A Scalable, Commodity Data Center Network Architecture. In *SIGCOMM'08*.
[10] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat. Hedera: dynamic flow scheduling for data center networks. In *NSDI'10*.
[11] M. Alizadeh, A. G. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan. Data center TCP (DCTCP). In *SIGCOMM'10*.
[12] M. Alizadeh, S. Yang, M. Sharif, S. Katti, N. McKeown, B. Prabhakar, and S. Shenker. pfabric: Minimal near-optimal datacenter transport. In *SIGCOMM'13*.
[13] T. Benson, A. Akella, and D. A. Maltz. Network traffic characteristics of data centers in the wild. In *IMC'10*.
[14] L. Chen, S. Hu, K. Chen, H. Wu, and D. H. K. Tsang. Towards minimal-delay deadline-driven data center tcp. In *Hotnets'13*.
[15] A. Curtis, J. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee. Devoflow: scaling flow management for high-performance networks. In *SIGCOMM'11*.
[16] J. Dean and S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. In *OSDI'04*.
[17] F. R. Dogar, T. Karagiannis, H. Ballani, and A. Rowstron. Decentralized Task-aware Scheduling for Data Center Networks. In *SIGCOMM'14*.
[18] C. Hong, M. Caesar, and P. Godfrey. Finishing flows quickly with preemptive scheduling. In *SIGCOMM'12*.
[19] V. Jeyakumar, M. Alizadeh, D. Mazieres, B. Prabhakar, C. Kim, and A. Greenberg. EyeQ: Practical Network Performance Isolation at the Edge. In *NSDI'13*.
[20] D. Katabi, M. Handley, and C. Rohrs. Congestion Control for High Bandwidth-Delay Product Networks. In *SIGCOMM'02*.
[21] G. Kumar, S. Kandula, P. Bodik, and I. Menache. Virtualizing traffic shapers for practical resource allocation. In *HotCloud'13*.
[22] A. Munir, I. A. Qazi, Z. A. Uzmi, A. Mushtaq, S. N. Ismail, M. S. Iqbal, and B. Khan. Minimizing Flow Completion Times in Data Centers. In *INFOCOM'13*.
[23] B. Vamanan, J. Hasan, and T. Vijaykumar. Deadline-aware datacenter tcp (d$^2$tcp). In *SIGCOMM'12*.
[24] C. Wilson, H. Ballani, T. Karagiannis, and A. Rowstron. Better never than late: Meeting deadlines in datacenter networks. In *SIGCOMM'11*.
[25] D. Zats, T. Das, P. Mohan, D. Borthakur, and R. Katz. Detail: Reducing the flow completion time tail in datacenter networks. In *SIGCOMM'12*.