PIAS: Practical Information-Agnostic Flow Scheduling for Commodity Data Centers

Wei Bai[1] Li Chen[1] Kai Chen[1] Dongsu Han[2] Chen Tian[3] Hao Wang[1]

[1]SING Group @ HKUST [2]KAIST [3]Nanjing University

***Abstract*— Many existing data center network (DCN) flow scheduling schemes that minimize flow completion times (FCT) assume prior knowledge of flows and custom switch functions, making them superior in performance but hard to implement in practice. By contrast, we seek to minimize FCT with no prior knowledge and existing commodity switch hardware.**

To this end, we present PIAS, a DCN flow scheduling mechanism that aims to minimize FCT by mimicking Shortest Job First (SJF) on the premise that flow size is not known *a priori*. At its heart, PIAS leverages multiple priority queues available in existing commodity switches to implement a Multiple Level Feedback Queue (MLFQ), in which a PIAS flow is gradually demoted from higher-priority queues to lower-priority queues based on the number of bytes it has sent. As a result, short flows are likely to be finished in the first few high-priority queues and thus be prioritized over long flows in general, which enables PIAS to emulate SJF without knowing flow sizes beforehand.

We have implemented a PIAS prototype and evaluated PIAS through both testbed experiments and ns-2 simulations. We show that PIAS is readily deployable with commodity switches and backward compatible with legacy TCP/IP stacks. Our evaluation results show that PIAS significantly outperforms existing information-agnostic schemes, for example, it reduces FCT by up to 50% compared to DCTCP [11] and L2DCT [34]; and it only has a 1.1% performance gap to an ideal information-aware scheme, pFabric [13], for short flows under a production DCN workload.

***Index Terms*— Data center networks, Flow scheduling**

I. Introduction

It has been a virtually unanimous consensus in the community that one of the most important objectives for data center network (DCN) transport designs is to minimize the flow completion times (FCT) [11]–[13, 27, 33, 34]. This is because many of today's cloud applications, such as web search, social networking, and retail recommendation, etc., have very demanding latency requirements, and even a small delay can directly affect application performance and degrade user experience [11, 34].

To minimize FCT, most previous proposals [13, 27, 33, 43] assume prior knowledge of accurate flow information, *e.g.*, flow sizes or deadlines, to achieve superior performance. For example, PDQ, pFabric and PASE [13, 27, 33] all assume flow size is known *a priori*, and attempt to approximate Shortest Job First (SJF, preemptive), which is the optimal scheduling discipline for minimizing the average FCT over a single link. In this paper, we question the validity of this assumption, and point out that, for many applications, such information is difficult to obtain, and may even be unavailable (§II). Existing transport layer solutions with this assumption are therefore very hard to implement in practice.

We take one step back and ask: without prior knowledge of flow size information, what is the best scheme that minimizes FCT with existing commodity switches?

Motivated by the above question, we set up our key design goals as follows:

- **Information-agnostic:** Our design must not assume *a priori* knowledge of flow size information being available from the applications.
- **FCT minimization:** The solution must be able to enforce a near-optimal information-agnostic flow schedule. It should minimize the average and tail FCT for latency-sensitive short flows, while not adversely affecting the FCT of long flows.
- **Readily-deployable:** The solution must work with existing commodity switches in DCNs and be backward compatible with legacy TCP/IP stacks.

When exploring possible solution spaces, we note that some existing approaches such as DCTCP, Timely, HULL, etc. [11, 12, 32, 34, 47] reduce FCT without relying on flow size information. They generally improve FCT by maintaining low queue occupancy through mechanisms like adaptive congestion control, ECN, pacing, etc. However, they do not provide a full answer to our question, because they mainly perform end-host based rate control which is ineffective for flow scheduling [13, 33].

In this paper, we answer the question with PIAS, a practical information-agnostic flow scheduling that minimizes the FCT in DCNs. In contrast to previous FCT minimization schemes [13, 27, 33] that emulate SJF by using prior knowledge of flow sizes, PIAS manages to mimic SJF with no prior information. At its heart, PIAS leverages multiple priority queues available in existing commodity switches to implement a Multiple Level Feedback Queue (MLFQ), in which a PIAS flow is gradually demoted from higher-priority queues to lower-priority queues based on the bytes it has sent during its lifetime. In this way, PIAS ensures in general that short flows are prioritized over long flows, effectively emulating SJF without knowing flow sizes beforehand.

Generally speaking, there are three components in the PIAS framework. A central entity collects traffic information reported from each end host periodically, calculates the demotion thresholds based on the aggregated traffic statistics from the entire network, and distributes the thresholds to all end

A subset of this work appears in USENIX NSDI'15 [14].

hosts. Then each host uses such thresholds to tag the traffic. And finally, the PIAS switches simply perform strict priority queueing based on the packet tags.

However, we face several concrete challenges before making PIAS truly effective. First, how to determine the demotion threshold for each queue of MLFQ? Second, as the traffic varies across both time and space, how to keep PIAS's performance in such a dynamic environment? Third, how to ensure PIAS's compatibility with legacy TCP/IP stacks in production DCNs?

For the first challenge, we derive a set of optimal demotion thresholds for MLFQ through solving a FCT minimization problem. We further show that the derived threshold set is robust to a reasonable range of traffic distributions. This enables PIAS to distribute the demotion thresholds to the end hosts for packet tagging while only performing strict priority queueing, a built-in function, in the PIAS switches.

For the second challenge, since one set of demotion thresholds only works the best for a certain range of traffic distributions, PIAS adjusts the thresholds to keep up with the traffic dynamics. However, the key problem is that a mismatch between thresholds and underlying traffic is inevitable. Once happens, short flows may be adversely affected by large ones in a queue, impacting on their latency. Inspired by ideas from ECN-based rate control [11], PIAS employs ECN to mitigate the mismatch problem. Our insight is that, by maintaining low queue occupancy, short flows always see small queues and thus will not be seriously delayed even if they are mistakenly placed in a queue with a long flow due to mismatched thresholds.

For the third challenge, we employ DCTCP-like transport at the PIAS end hosts and find that PIAS interacts favorably with DCTCP or other legacy TCP protocols with ECN enabled. A potential problem is that many concurrent short flows may starve a coexisting long flow, triggering TCP timeouts and degrading application performance. We measure the extent of starvation on our testbed with a realistic workload and analyze possible solutions. We ensure that all mechanisms in PIAS can be implemented by a shim layer over NIC without touching the TCP stack.

We have implemented a PIAS prototype (§IV). On the end host, we implement PIAS as a kernel module in Linux, which resides between the Network Interface Card (NIC) driver and the TCP/IP stack as a shim layer. It does not touch any TCP/IP implementation that natively supports various OS versions. In virtualized environments, PIAS can also support virtual machines well by residing in hypervisor (or Dom 0). At the switch, PIAS only needs to enable priority queues and ECN which are both built-in functions readily supported by existing commodity switch hardware.

We evaluate PIAS on a small-scale testbed with 16 Dell servers and a commodity Pronto-3295 Gigabit Ethernet switch (Broadcom BCM#56538). In our experiments, we find that PIAS reduces the average FCT for short flows by $\sim$29-49% and $\sim$18-34% compared to DCTCP under two realistic DCN traffic patterns. It also improves the query performance by $\sim$28-30% in a Memcached [6] application (§V-A). We further dig into different design components of PIAS such as queues, optimal demotion threshold setting, ECN, and demonstrate the effectiveness of each of their contributions to PIAS's performance (§V-B).

To complement our small-scale testbed experiments, we further conduct large-scale simulations in a simulated 10/40G network using ns-2 [9]. In our simulations, we show that PIAS outperforms all existing information-agnostic solutions under realistic DCN workloads, reducing the average FCT for short flows by up to 50% compared to DCTCP and L2DCT. In addition, our results show that PIAS, as a readily-deployable information-agonistic scheme, also delivers a comparable performance to a clean-slate information-aware design, pFabric [13], in certain scenarios. For example, there is only a 1.1% gap to pFabric for short flows in a data mining workload [25] (§V-C).

To make our work easy to reproduce, we made our implementation and evaluation scripts available online at: `http://sing.cse.ust.hk/projects/PIAS`.

II. Motivation

To motivate our design, we introduce a few cases in which the accurate flow size information is hard to obtain, or simply not available.

HTTP chunked transfer: Chunked transfer has been supported since HTTP 1.1 [23], where dynamically generated content is transferred during the generation process. This mode is widely used by datacenter applications. For example, applications can use chunked transfer to dump database content into OpenStack Object Storage [8]. In chunked transfer, a flow generally consists of multiple chunks, and the total flow size is not available at the start of the transmission.

Database query response: Query response in database systems, such as Microsoft SQL Server [7], is another example. Typically, SQL servers send partial query results as they are created, instead of buffering the result until the end of the query execution process [7]. The flow size again is not available at the start of a flow.

Stream processing: Stream processing systems are currently gaining popularity. In Apache Storm [2], after the master node distributes tasks to worker nodes, workers will analyze the tasks and pre-establish persistent connections with related worker nodes. During the data processing, data tuples completed in one node are continuously delivered to the next node in the stream processing chain. The amount of data to be processed is unknown until the stream finishes.

Practical limitations: We note that there are certain cases where the flow size information can be obtained or inferred. For example, in Hadoop [4] the mapper will first write the intermediate data to disk before the corresponding reducer starts to fetch the data, thus the flow size can be obtained in advance [35]. Even so, practical implementation issues are still prohibitive. First, we need to patch all modules in every application that generate network traffic, which is a burden for applications programmers and/or network operators. Second, current operating systems lack appropriate interface for delivering the flow size information to the transport layer. Thus, kernel modifications are also required.

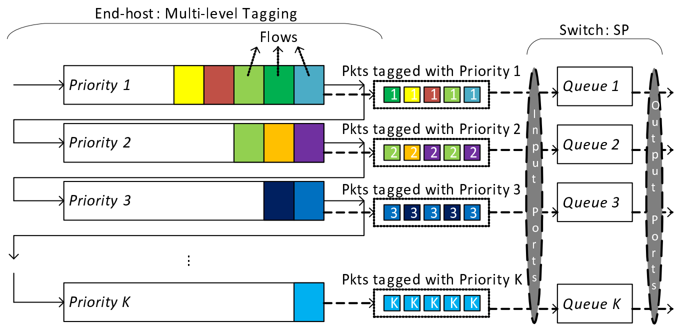

End-host : Multi-level Tagging
Flows
Priority 1
Priority 2
Priority 3
Priority K
Pkts tagged with Priority 1
Pkts tagged with Priority 2
Pkts tagged with Priority 3
Pkts tagged with Priority K
Switch: SP
Input Ports
Queue 1
Queue 2
Queue 3
Queue K
Output Ports

Fig. 1: PIAS overview

III. The PIAS Design

A. Design Rationale

Compared to previous solutions [13, 27, 33] that emulate SJF based on prior knowledge of flow sizes, PIAS distinguishes itself by emulating SJF with no prior information. At its core, PIAS exploits multiple priority queues available in commodity switches and implements a MLFQ, in which a PIAS flow is demoted from higher-priority queues to lower-priority queues dynamically according to its bytes sent. Through this way, PIAS enables short flows to finish in the first few priority queues, and thus in general prioritizes them over long flows, effectively mimicking SJF without knowing the flow sizes.

We note that scheduling with no prior knowledge is known as non-clairvoyant scheduling [30]. Least Attained Service (LAS) is one of the best known algorithms that minimize the average FCT in this case [39]. LAS tries to approximate SJF by guessing the remaining service time of a job based on the service it has attained so far. LAS is especially effective in DCN environments where traffic usually exhibits long-tail distribution—most flows are short and a small percent are very large [11, 25], and LAS favors short flows.

PIAS is partially inspired by LAS. However, we note that directly enabling LAS at the switch requires us to compare the amount of bytes transferred for each flow, which is not supported in existing commodity switches. Furthermore, although DCN traffic is generally long-tailed, it varies across both time and space, and on some switch ports the distribution may temporarily not be so. Blindly using LAS will exacerbate the problem when multiple long flows coexist. This is because pure LAS needs to compare and prioritize between any two flows, when some long flows meet a longer flow, it is possible that they can collectively cause the longer one to starve.

To this end, PIAS leverages multiple priority queues available in existing commodity switches (typically 4–8 queues per port [13]) to implement a MLFQ (see Figure 1). Packets in different queues of MLFQ are scheduled with strict priority, while packets in the same queue are scheduled based on FIFO. In a flow's lifetime, it is demoted dynamically from ith queue down to the $(i+1)$th queue after transmitting more bytes than queue i's demotion threshold, until it enters the last queue. To further prevent switches from maintaining the per-flow state, PIAS distributes packet priority tagging (indicating a flow's sent size) to end hosts, allowing the PIAS switches to perform strict priority queueing only, which is already a built-in function in today's commodity switches.

By implementing MLFQ, PIAS gains two benefits. First, it prioritizes short flows over large ones because short flows are more likely to finish in the first few higher priority queues while large flows are eventually demoted to lower priority queues. This effectively enables PIAS to approximate SJF scheduling that optimizes average FCT while being readily implementable with existing switch hardware. Second, it allows large flows that are demoted to the same low priority queues to share the link fairly. This helps to minimize the response time of long flows, mitigating the starvation problem.

However, there are several concrete challenges to consider in order to make PIAS truly effective. First, how to determine the demotion threshold for each queue of MLFQ to minimize the FCT? Second, as DCN traffic varies across both time and space, how to make PIAS perform efficiently and stably in such a dynamic environment? Third, how to ensure PIAS's compatibility with legacy TCP/IP stacks in production DCNs? Next, we explain the details of the mechanism we design to address all these challenges.

B. Detailed Mechanisms

At a high level, PIAS's main mechanisms include packet tagging, switch design, and rate control.

1) Packet Tagging at End-hosts: PIAS performs packet tagging at each end host as shown in Figure 1. There are K priorities $P_i, 1 \le i \le K$ and $(K-1)$ demotion thresholds $\alpha_j, 1 \le j \le K-1$. We assume $P_1 > P_2 ... > P_K$ and $\alpha_1 \le \alpha_2 ... \le \alpha_{K-1}$.

At the end host, when a new flow is initialized, its packets will be tagged with the highest priority P_1, giving it the highest priority in the network. As more bytes are sent, the packets of this flow will be tagged with decreasing priorities P_j $(2 \le j \le K)$ and receive decreasing priorities in the network. The threshold to demote priority from P_{j-1} to P_j is α_{j-1}.

One challenge is to determine the demotion threshold for each priority to minimize the average FCT. By solving a FCT minimization problem, we derive a set of analytical solutions for optimal demotion thresholds (details in §III-C). Note that in PIAS we calculate the thresholds based on traffic information from the entire DCN and distribute the same threshold setting to all the end hosts. Our experiments and analysis show that such threshold setting is effective and also robust to a certain range of traffic variations (§V). This is a key reason we can decouple packet tagging from switches to end hosts while still maintaining good performance, which relieves the PIAS switches of having to keep the per-flow state.

As traffic changes over time, PIAS needs to adjust the demotion thresholds accordingly. To keep track of traffic variations, each end host can periodically report its local traffic information to a central entity for statistics, and there are many existing techniques available for this purpose [46]. However, historical traffic cannot predict the future perfectly, and traffic distribution also differs across different links. Mismatches between threshold setting and underlying traffic are inevitable, which can hurt latency sensitive short flows.

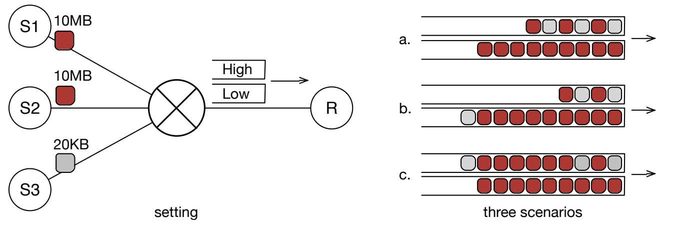


Fig. 2: Illustration example: (a) threshold right; (b) threshold too small, packets of short flow get delayed by long flow after prematurely demoted to the low priority queue; (c) threshold too large, packets of large flow stay too long in the high priority queue, affecting short flow.

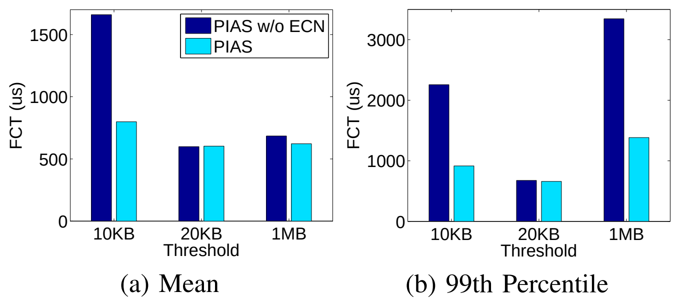


Fig. 3: Completion times of 20KB short flows.

Therefore, mitigating the impact of the mismatch is a must for PIAS to operate in the highly dynamic DCNs. Our solution, as shown subsequently, is to employ ECN.

2) Switch Design: The PIAS switches enable the following two basic mechanisms, which are built-in functions for existing commodity switches [33].

- Priority scheduling: Packets are dequeued based on their priorities strictly when a fabric port is idle.
- ECN marking: The arriving packet is marked with Congestion Experienced (CE) if the instant buffer occupancy is larger than the marking threshold.

With priority scheduling at the switches and packet tagging at the end hosts, PIAS performs MLFQ-based flow scheduling using stateless switches in the network. Packets with different priority tags are classified into different priority queues. When the link is idle, the head packet from the highest non-empty priority queue is transmitted.

We choose priority scheduling over weighted fair queueing (WFQ) [21] for two reasons. First, strict priority queueing provides better in-network prioritization than WFQ and thus can achieve lower average FCT. Second, packets from a PIAS flow will be placed and scheduled in multiple queues, with WFQ, it may cause packet out-of-order problem as a latter packet has a chance to dequeue before an earlier packet, which degrades TCP performance. However, with strict priority queueing, PIAS is free of packet reordering, because a flow is always demoted from a higher priority queue to a lower priority queue, and an earlier packet is guaranteed to dequeue before a latter packet at each hop. Compared with priority scheduling, one advantage of WFQ is that it can potentially avoid starvation for long-lived flows. We quantify the starvation problem using testbed experiments subsequently and find that, even with priority scheduling, PIAS is not adversely affected by starvation.

Our intention to employ ECN is to mitigate the effect of the mismatch between the demotion thresholds and the traffic distribution. We use a simple example to illustrate the problem and the effectiveness of our solution. We connect 4 servers to a Gigabit switch as in Figure 2. One server is receiver (R) and the other three are senders (S1, S2 and S3). In our experiment, the receiver R continuously fetches 10MB data from S1 and S2, and 20KB data from S3. We configure the strict priority queueing with 2 queues on the switch egress port to R. Since there are two priorities, we only need to have one demotion threshold from the high priority queue to the low priority queue. Clearly, in this case the optimal demotion threshold should be 20KB.

We intentionally apply three different demotion thresholds 10KB, 20KB and 1MB respectively, and measure the FCT of the 20KB short flows. Figure 3 shows the results of PIAS and PIAS without ECN. When the threshold is 20KB (right), both PIAS and PIAS without ECN achieve an ideal FCT. However, with a larger threshold (1MB) or a smaller threshold (10KB), PIAS exhibits pronounced advantages over PIAS without ECN at both the average and 99th percentile. This is because, 1) if the threshold is too small, packets of short flows prematurely enter the low priority queue and experience queueing delay behind the long flows (see the scenario in Figure 2 (b)); 2) if the threshold is too large, packets of long flows over-stay in the high priority queue, thus also affecting the latency of short flows (see scenario in Figure 2 (c)).

By employing ECN, we can keep low buffer occupancy and minimize the impact of long flows on short flows, which makes PIAS more robust to the mismatch between the demotion thresholds and traffic distribution.

3) Rate Control: PIAS employs DCTCP [11] as end host transport, and other legacy TCP protocols with ECN enabled (*e.g.*, ECN* [44] and DCQCN [47]) can also be integrated into PIAS. We require PIAS to interact smoothly with the legacy TCP stack. One key issue is to handle flow starvation: when packets of a large flow get starved in a low priority queue for long time, this may trigger TCP timeouts and retransmissions. The frequent flow starvation may disturb the transport layer and degrade application performance. For example, a TCP connection which is starved for long time may be terminated unexpectedly.

To address the problem, we first note that PIAS can well mitigate the starvation between long flows, because two long flows in the same low priority queue will fairly share the link in a FIFO manner. In this way, PIAS minimizes the response time of each long flow, effectively eliminating TCP timeouts.

However, it is still possible that many concurrent short flows will starve a long flow, triggering its TCP timeouts.To quantify this, we run the web search benchmark traffic [11] (as shown in Figure 5) at 0.8 load in our 1G testbed, which has 16 servers connected to a Gigabit switch. We set RTOmin to 10ms and allocate 8 priority queues for PIAS. We enable both ECN and dynamic buffer management in our switch. In such setting, TCP timeouts, if happen, will be mainly caused by starvation rather than packet drops. This experiment consists of 5,000 flows, around 5.7 million MTU-sized packets. We measure the number of TCP timeouts to quantify the extent of starvation. We find that there are only 200 timeout events and 31 two consecutive timeout events in total. No TCP connection is terminated unexpectedly. The result indicates that, even at a high load, starvation is not common and will not degrade application performance adversely. We believe one possible reason is that the per-port ECN we used (see §IV-A.2) may mitigate starvation by pushing back high priority flows when many packets from low priority long flows get starved. Another possible solution for handling flow starvation is treating a long-term starved flow as a new flow. For example, if a flow experiences two consecutive timeouts, we set its bytes sent back to zero. This ensures that a long flow can always make progress after timeouts. Note that the implementation of the above mechanism can be integrated to our packet tagging module without any changes to the networking stack.

4) *Discussion:*

We further discuss two potential issues.

Local decision: The key idea of PIAS is to emulate SJF which is optimal to minimize average FCT over a single link. However, there does not exist an optimal scheduling policy to schedule flows over an entire DCN with multiple links [13]. In this sense, similar to pFabric [13], PIAS also makes switch local decisions. This approach in theory may lead to some performance loss over the fabric [33]. For example, when a flow traverses multiple hops and gets dropped at the last hop, it causes bandwidth to be wasted on the upstream links that could otherwise have been used to schedule other flows. We note that some existing solutions [27, 33] leverage arbitration, where a common network entity allocates rates to each flow based on global network visibility, to address this problem. However, they are hard to implement because they require either non-trivial switch changes [27] or a complex control plane [33], which is against our design goal. Furthermore, some arbitration mechanisms [33] implicitly assume that each flow uses a single path. This assumption does not hold when network operators use advanced per-packet/flowlet/flowcell load balancing schemes [10, 18, 26]. Fortunately, local-decision based solutions maintain very good performance for most scenarios [13] and only experience performance loss at extremely high loads, *e.g.*, over 90% [33]. However, most DCNs operate at moderate loads, *e.g.*, 30% [16]. Our ns-2 simulation (§V-C) with production DCN traffic further confirms that PIAS works well in practice.

Demotion threshold updating: By employing ECN, PIAS can effectively handle the mismatch between the demotion thresholds and traffic distribution (see §V-B). This suggests that we do not need to frequently change our demotion thresholds which may be an overhead. In this paper, we simply assume the demotion thresholds are updated periodically according to the network scale (which decides the time for information collection and distribution) and leave dynamic threshold updates as future work.

C. Optimizing Demotion Thresholds

In this section, we describe our formulation to derive the optimal demotion thresholds for minimizing the average FCT. We leverage existing optimization software to derive the optimal thresholds numerically for *any* given load and flow size distribution. We find that the demotion thresholds depend on both load and flow size distribution. As flow size distribution and load change across both time and space, ideally, one should use different thresholds for different links at different times. However, in practice, it is quite challenging to obtain such fine-grained link level traffic information across the entire DCN. Hence, we use the overall flow size distribution and load measured in the entire DCN as an estimate to derive a common set of demotion thresholds for all end hosts. We note that this approach is not optimal and there is room for improvement. However, it is more practical and provides considerable gains, as shown in our evaluation (§V).

Problem formulation: We assume there are K priority queues P_i, $1{\le}i{\le}K$, where P_1 has the highest priority. We denote the threshold for demoting the priority from $i{-}1$ to i as α_{i-1}, $2{\le}i{\le}K$. We define $\alpha_K{=}\infty$, so that the largest flows are all in this queue, and $\alpha_0{=}0$.

Denote the cumulative density function of flow size distribution as $F(x)$, thus $F(x)$ is the probability that a flow size is no larger than x. Let $\theta_i{=}F(\alpha_i){-}F(\alpha_{i-1})$, the percentage of flows with sizes in $[\alpha_{i-1},\alpha_i)$. Let L_i denote the number of packets per flow brings in queue P_i for $i{=}1,...,K$. Thus, $E[L_i]{\le}(\alpha_i{-}\alpha_{i-1})\theta_i$. Denote flow arrival rate as λ, then the packet arrival rate to queue i is $\lambda_i{=}\lambda E[L_i]$.

The service rate for a queue depends on whether the queues with higher priorities are all empty. Thus, P_1 (highest priority) has capacity $\mu_1{=}\mu$ where μ is the service rate of the link. The fraction of time that queue 1 is idle is $(1{-}\rho_1)$ where $\rho_i{=}\lambda_i/\mu_i$ is simply the utilization of queue P_i. Thus, the service rate of P_2 is $\mu_2{=}(1{-}\rho_1)\mu$ since its service rate is μ (the full link capacity) given that P_1 is empty. Thus, we have $\mu_i{=}\Pi_{j=0}^{i-1}(1{-}\rho_j)\mu$, with $\rho_0{=}0$. Thus, $T_i{=}1/(\mu_i{-}\lambda_i)$ which is the average delay of queue i assuming M/M/1 queues[1].

For a flow with size in $[\alpha_{i-1},\alpha_i)$, it experiences the delays in different priority queues up to the i-th queue. Denote T_i as the average time spent in the i-th queue. Let $i_{max}(x)$ be the index of the smallest demotion threshold larger than x. So the average FCT for a flow with size x, $T(x)$, is upper-bounded by: $\sum_{i=1}^{i_{max}(x)} T_i$.

Thus we have an optimization problem of choosing an optimal set of thresholds $\{\alpha_i\}$ to minimize the average FCT of flows on this bottleneck link. Using $\theta_i{=}F(\alpha_i){-}F(\alpha_{i-1})$ to equivalently express α_i, we can formulate the problem as:

$$\min_{\{\theta_i\}} \quad \mathcal{T}{=}\sum_{l=1}^{K}(\theta_l\sum_{m=1}^{l}T_m){=}\sum_{l=1}^{K}(T_l\sum_{m=l}^{K}\theta_m) \qquad (1)$$

$$\text{subject to} \quad \theta_i{\ge}0, i{=}1,...,K{-}1$$

The solution of this problem is $\{\theta_i^{opt}\}$, from which we can retrieve the thresholds $\{\alpha_i^{opt}\}$ with $\alpha_i^{opt}{=}F^{-1}(\sum_{j=1}^{i}\theta_j)$, where $F^{-1}(\cdot)$ is the inverse of $F(\cdot)$.

[1] We use M/M/1 queues to simplify the analysis, and the robustness of this formulation is verified in §V experimentally.

Solution method: We have products of the variables (θ_i) both in numerators and denominators of the objective. Since the number of variables is small (the number of queues $\leq$8), the optimization problem can be solved numerically and quickly with existing solvers. We use global optimization toolbox available in MATLAB [20], which searches for the optimal solution using genetic algorithm. For 2, 4, 8-queue cases, the problem is solved in $18.12s$, $87.92s$, $321.78s$ respectively on a server in our testbed (described in §V). Since the flow size distribution is collected over a long period of time[2], the time for calculating the thresholds is negligible in comparison.

In summary, given flow arrival rate λ[3] and flow size distribution $F(\cdot)$, we can compute the thresholds relatively quickly, and enforce them at the end-hosts.

IV. Implementation and Testbed Setup

A. PIAS Implementation

We have implemented a prototype of PIAS. We now describe each component of our prototype in detail.

1) Packet Tagging: The packet tagging module is responsible for maintaining per-flow state and marking packets with priority at end hosts. We implement it as a kernel module in Linux. The packet tagging module resides between the TCP/IP stack and Linux TC, which consists of three components: a NETFILTER [5] hook, a hash based flow table, and a packet modifier.

The operations are as follows: 1) the NETFILTER hook intercepts all outgoing packets using the LOCAL_OUT hook and directs them to the flow table. 2) Each flow in the flow table is identified by the 5-tuple: src/dst IPs, src/dst ports and protocol. When a packet comes in, we identify the flow it belongs to (or create a new entry) and increment the amount of bytes sent. 3) Based on the flow information, the packet modifier sets the packet's priority by modifying the DSCP field in the IP header to the corresponding value.

In certain cases, applications may build persistent TCP connections to keep delivering request-response short messages for a long time. In such scenario, PIAS should provide message-level rather than connection-level prioritization. Otherwise, these persistent connections will eventually be assigned to the lowest priority due to the large cumulative size of bytes sent. To address this challenge, we monitor TCP send buffer instead of modifying applications [15] to get messages semantics. In our implementation, we add a JPROBE hook to TCP_SENDMSG function and record TCP_SENDMSG called time for each TCP flow. When TCP_SENDMSG is called for a flow, if the time gap since last function call is larger than a threshold (500us in our experiments) and the TCP send buffer is empty, we regard this as the beginning of a new message and reset bytes sent back to 0. We find this simple solution work well in the testbed experiments.

[2]The shortest period to collect enough flow information to form an accurate and reliable flow size distribution is an interesting research problem, which we leave as future work. Previous reported distributions [11, 17, 31] are all collected over periods of at least days.

[3]The traffic load used in §V can be calculated by $\lambda{\cdot}E[M]$, where $E[M]$ is the average flow size given $F(\cdot)$.

Offloading techniques like large segmentation offload (LSO) may degrade the accuracy of packet tagging. With LSO, the packet tagging module may not be able to set the right DSCP value for each individual MTU-sized packet within a large segment. To quantify this, we sample more than 230,000 TCP segments with payload data in our 1G testbed and find that the average segment size is only 7,220 Bytes. This introduces little impact on packet tagging. We attribute this to the small window size in DCN environment which has small bandwidth-delay product and large number of concurrent connections. We expect that the final implementation solution for packet tagging should be in NIC hardware to permanently avoid this interference.

To quantify system overhead introduced by the PIAS packet tagging module, we installed it on a Dell PowerEdge R320 server with a Intel 82599EB 10GbE NIC and measured CPU usage. LSO is enabled in this experiment. We started 8 TCP long flows and achieved ~9.4Gbps goodput. The extra CPU usage introduced by PIAS is $< 1\%$ compared with the case where the PIAS packet tagging module is not enabled.

In the near future, data center bandwidth can reach 100Gbps. In such high speed networks, people may offload network stacks to hardware to reduce CPU overhead and achieve low latency. Despite the above software implementation, the PIAS packet tagging module can also be implemented in hardware, such as programmable NICs and FPGA.

2) Switch Configuration: We enforce strict priority queues at the switches and classify packets based on the DSCP field. Similar to previous work [11, 44], we use ECN marking based on the instant queue lengths with a single marking threshold. In addition to the switch queueing delay in the network, the sender's NIC also introduces latency because it is actually the first contention point of the fabric [13, 29]. Hardware and software queues at the end hosts can introduce large queueing delay, which might severely degrade the application performance [28, 45]. To solve this problem, our software solution hooks into the TX datapath at POST_ROUTING and rate-limits outgoing traffic at the line rate. Then, we perform ECN marking and priority queueing at the end host as well as the switches.

Per-queue vs per-port ECN marking: We observe that some of today's commodity switching chips offer multiple ways to configure ECN marking when configured to use multiple queues per port. For example, our Broadcom BCM#56538-based switch allows us to enable either per-queue ECN marking or per-port ECN marking. In per-queue ECN marking, each queue has its own marking threshold and performs ECN marking independently to other queues. In per-port ECN marking, each port is assigned a single marking threshold and marks packets when the sum of all queue sizes belong to the port exceeds the marking threshold.

Per-queue ECN is widely used in many DCN transport protocols [11, 33, 41], however, we find it has limitation when supporting multiple queues. Each queue requires a moderate ECN marking threshold h to fully utilize the link independently (*e.g.*, h=20 packets for 1G and 65 packets for 10G in DCTCP [11]). Thus, supporting multiple queues

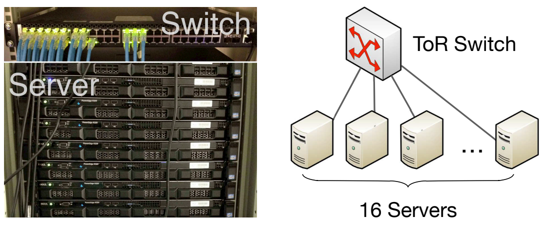


Fig. 4: Testbed Topology

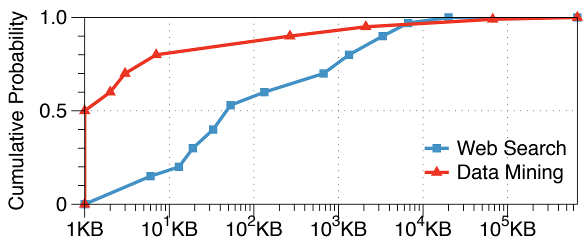


Fig. 5: Traffic distributions used for evaluation.

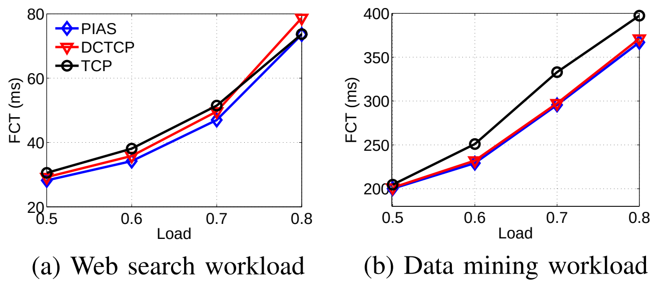


Fig. 6: Overall average flow completion times for both workloads.

may require the shared memory be at least multiple times (*e.g.*, 8) the marking threshold, which is not affordable for most shallow buffered commodity switches. For example, our Pronto-3295 switch has 4MB ($\approx$2667 packets) memory shared by 384 queues (48x 1G ports with 8 queues per port). If we set h=20 packets as suggested above, we need over 11MB memory in the worst case, otherwise when the traffic is bursty, the shallow buffer may overflow before ECN takes effect.

Per-port ECN, to the best of our knowledge, has rarely been exploited in recent DCN transport designs. Although per-port ECN marking cannot provide ideal isolation among queues as per-queue ECN marking , it can provide much better burst tolerance and support a larger number of queues in shallow buffered switches. Moreover, per-port ECN marking can potentially mitigate the starvation problem. It can push back high priority flows when many packets of low priority flows get queued in the switch. Therefore, we use per-port ECN marking.

3) Rate Control: We use Linux 3.18.11 kernel that supports DCTCP congestion control algorithm. We further observe an undesirable interaction between the DCTCP implementation and our switch. The DCTCP implementation does not set the ECN-capable (ECT) codepoint on TCP SYN packets and retransmitted packets, following the ECN standard [40]. However, our switch drops any non-ECT packets from ECN-enabled queues, when the instant queue length is larger than the ECN marking threshold. This problem severely degrades the TCP performance [44]. To address this problem, we set ECT on every TCP packet at the packet modifier.

B. Testbed Setup

We built a small testbed that consists of 16 servers connected to a Pronto 3295 48-port Gigabit Ethernet switch with 4MB shared memory, as shown in Figure 4. Our switch supports ECN and strict priority queuing with at most 8 class of service queues [1]. Each server is a Dell PowerEdge R320 with a 4-core Intel E5-1410 2.8GHz CPU, 8G memory, a 500GB hard disk, and a Broadcom BCM5719 NetXtreme Gigabit Ethernet NIC. Each server runs Debian 7.8-64bit with Linux 3.18.11 kernel. By default, advanced NIC offload mechanisms are enabled to reduce the CPU overhead. The base round-trip time (RTT) of our testbed is around 135us.

In addition, we have also built a smaller 10G testbed for measuring the end host queueing delay in high speed network. We connect three servers to the same switch (Pronto 3295 has four 10GbE ports). Each server is equipped with an Intel 82599EB 10GbE NIC.

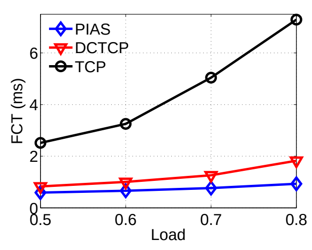
V. Evaluation

We evaluate PIAS using a combination of testbed experiments and large-scale ns-2 simulations. Our evaluation centers around four key questions:

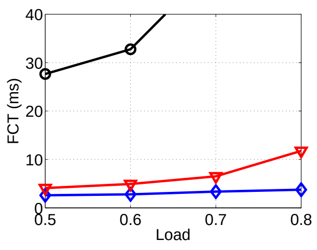
- **How does PIAS perform in practice?** Using realistic workloads in our testbed experiments, we show that PIAS reduces the average FCT of short flows by ~29-49% with the web search workload [11] and ~18-34% with the data mining workload [25] compared to DCTCP. In an application benchmark with Memcached [6], we show that PIAS achieves ~28-30% lower average query completion time than DCTCP.
- **How effective are individual design components of PIAS**, and how sensitive is PIAS to parameter settings? We show that PIAS achieves reasonable performance even with two queues. We also demonstrate that ECN is effective in mitigating the harmful effect of a mismatch between the demotion thresholds and traffic, but PIAS performs the best with the optimal threshold setting.
- **Does PIAS work well even in large datacenters?** Using large-scale ns-2 simulations, we show that PIAS scales to multi-hop topologies and performs best among all information-agnostic schemes (DCTCP [11], L2DCT [34], and LAS [39]). PIAS shows a 1.1% performance (measured in average FCT) gap from pFabric [13], an idealized information-aware scheme, for short flows in the data mining workload.

A. Testbed Experiments

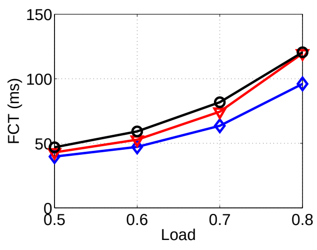
Setting: PIAS uses 8 priority queues by default and enables per-port ECN marking as discussed in §IV. Given the based RTT is only 135us, we set the ECN marking threshold to 20KB. As many works [11, 42] suggest, we set TCP RTOmin to 10ms. The TCP initial window is 10 packets. Our MLFQ demotion thresholds are derived as described in §III.



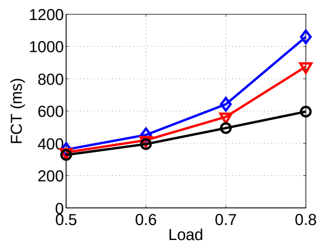
(a) (0,100KB]: Avg



(b) (0,100KB]: 99th Percentile

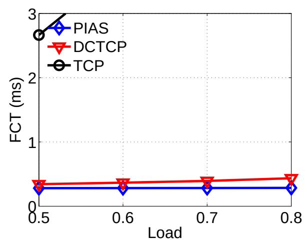


(c) (100KB,10MB]: Avg

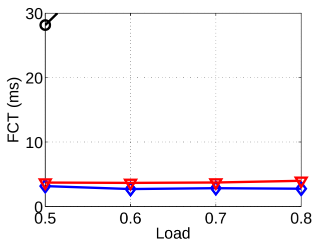


(d) (10MB,∞): Avg

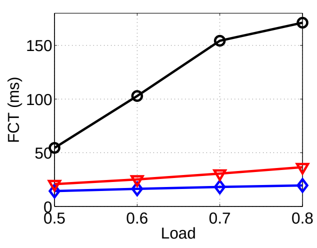
Fig. 7: Web search workload: FCT across different flow sizes. TCP's performance is outside the plotted range of (b)



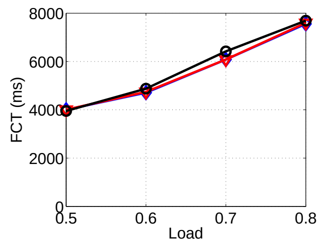
(a) (0,100KB]: Avg



(b) (0,100KB]: 99th Percentile



(c) (100KB,10MB]: Avg



(d) (10MB,∞): Avg

Fig. 8: Data mining workload: FCT across different flow sizes. TCP's performance is outside the plotted range of (a) and (b)

We use two realistic workloads, a web search workload [11] and a data mining workload [25] from production datacenters. Their overall flow size distributions are shown in Figure 5. We also evaluate PIAS using an application benchmark with Memcached [6].

Results with realistic workloads: For this experiment, we develop a client/server model[4] to generate dynamic traffic according to realistic workloads and measure the FCT on application layer. The client application, running on 1 machine, initially opens 5 persistent TCP connections to each of the rest 15 machines. During the experiment, the client application periodically generates requests through available connections (if no connection is available, the client application will establish a new persistent connection) to the other machines to fetch data. The server applications, running on 15 other machines, respond with requested data. Hence, a TCP connection can carry multiple flows (messages). The requests are generated based on a Poisson process. We evaluate the performance of PIAS, DCTCP and TCP, while varying the network loads from 0.5 to 0.8. We run 10000 flows for each setting.

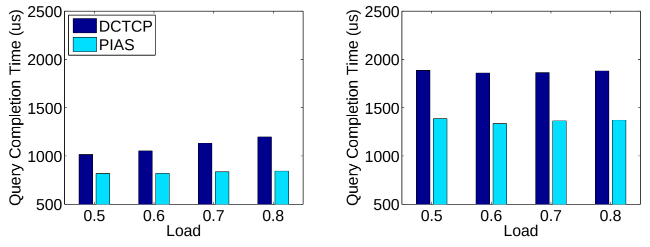
Figure 6 gives the overall average FCT for the web search workload and the data mining workload at various loads. In general, PIAS delivers the best performance. In the web search workload, the overall average FCT with PIAS is up to 6.5% lower compared to DCTCP and 10.2% lower compared to TCP. In the data mining workload, PIAS reduces the overall average FCT by up to 1.2% and 11.2% compared to DCTCP and TCP, respectively.

We further break down FCT across different flow sizes. Figure 7 and Figure 8 show the FCT across small (0,100KB] (a, b), medium (100KB,10MB] (c), and large (10MB,∞) (d) flows, respectively; for the web search and data mining workloads, respectively.

We make the following three observations: First, for both workloads, PIAS achieves the best performance in both the average and 99th percentile FCTs of small flows. Compared to DCTCP, PIAS reduces the average FCT of small flows by ~29-49% for web search workload and ~18-34% for data mining workload. The improvement of PIAS over DCTCP in the 99th percentile FCT of short flows is also obvious: ~37-62% for the web search workload and ~15-31% for the data mining workload. Second, PIAS also provides the best performance in medium flows. It achieves up to 20% lower average FCT of medium flows than DCTCP in the web search workload. Third, PIAS does not severely penalize the large flows. For example, from Figure 7 (d) and Figure 8 (d), we can see that for the data mining workload PIAS is comparable or slightly better than TCP and DCTCP, while for the web search workload it is worse than DCTCP by up to 21%. This is expected because PIAS prioritizes short flows over long flows and ~60% of all bytes in the web search workload are from flows smaller than 10MB. Note that the performance gap for large flows would not affect the overall average FCT since datacenter workloads are dominated by small and medium flows.

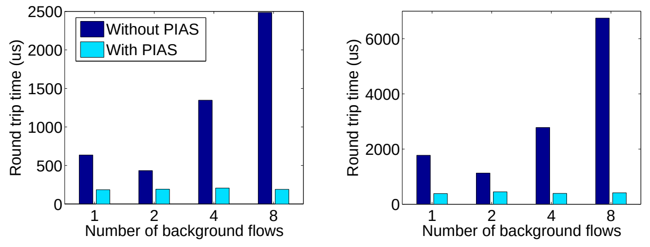
Results with the Memcached application: To assess how PIAS improves the performance of latency-sensitive applications, we build a Memcached [6] cluster with 16 machines. One machine is used as a client and the other 15 are used as servers to emulate a partition/aggregate soft-real-time service [11, 43]. We pre-populate server instances with 4B-key, 1KB-value pairs. The client sends a GET query to all 15 servers and each server responds with a 1KB value. A query

[4]https://github.com/HKUST-SING/TrafficGenerator



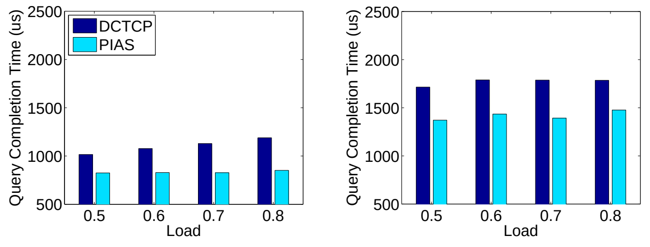
(a) Mean (b) 99th Percentile

Fig. 9: Query completion time at 20 qps



(a) Mean (b) 99th Percentile

Fig. 11: RTT with background flows



(a) Mean (b) 99th Percentile

Fig. 10: Query completion time at 40 qps

is completed only when the client receives all the responses from the servers. We measure the query completion time as the application performance metric. Since a 1KB response can be transmitted within one RTT, the query completion time is mainly determined by the tail queueing delay. The base query completion time is around 650us in our testbed. We also generate background traffic, a mix of mice flows and elephant flows following the distribution of the web search workload [11]. We use queries per second, or *qps*, to denote the application load. We vary the load of the background traffic from 0.5 to 0.8 and compare the performance of PIAS with that of DCTCP.

Figure 9 and Figure 10 show the results of the query completion time at 20 and 40 qps loads respectively. Since we enable both dynamic buffer management and ECN at the switch, none of queries suffers from TCP timeout. With the increase in background traffic load, the average query completion time of DCTCP also increases (1016–1189us at 40qps and 1014–1198us at 20qps). By contrast, PIAS maintains a relatively stable performance. At 0.8 load, PIAS can achieve $\sim$28-30% lower average query completion times than those of DCTCP. Moreover, PIAS also reduces the 99th percentile query completion time by $\sim$20-27%. In summary, PIAS can effectively improve the performance of the Memcached application by reducing the queueing delay of short flows.

End host queueing delay: The above experiments mainly focus on network switching nodes. PIAS extends its switch design to the end hosts as the sender's NIC is actually the first contention point of the fabric [13, 29].

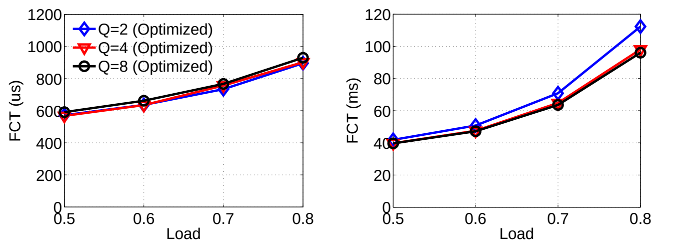
To quantify this, we conduct an experiment in our 10G setting with three servers (one sender and two receivers). We start several (1 to 8) long-lived TCP flows from the sender to a receiver. Then we measure RTT from the sender to the other receiver by sending ICMP `ping` packets. Without PIAS, `ping` packets could experience up to 6748us queueing delay with 8 background flows. Then we deploy a 2-queue PIAS end host scheduling module (as described in §IV-A.2) with a threshold of 100KB. Each ICMP packet is identified as a new flow by PIAS. We measure the RTTs with PIAS and compare them with the results without PIAS in Figure 11. In general, PIAS can significantly reduce the average RTT to $\sim$200us and ensure that the 99th percentile RTT is smaller than 450us. Note that the PIAS scheduling module does not affect network utilization and large flows still maintain more than 9Gbps goodput during the experiment. Since we enable LSO to reduce CPU overhead, it is difficult for us to achieve fine-grained transmission control and some delay may still exist in NIC's transmission queues. We believe there is still room to improve by offloading the scheduling to NIC hardware [38].

B. PIAS Deep Dive

In this section, we conduct a series of targeted experiments to answer the following three questions:

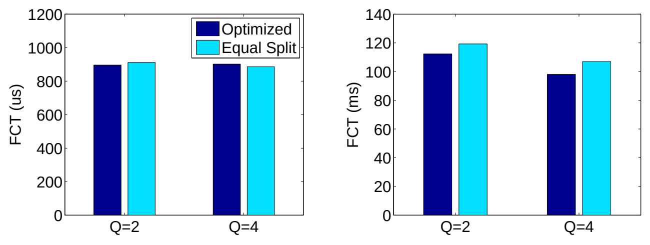
- **How sensitive is PIAS to the number of queues available?** Network operators may reserve some queues for other usage while some commodity switches [3] only support 2 priority queues. We find that, even with only 2 queues, PIAS still effectively reduces the FCT of short flows. However, in general, more queues can further improve PIAS's overall performance.
- **How effective is ECN in mitigating the mismatch?** ECN is integrated into PIAS to mitigate the mismatch between the demotion thresholds and traffic distribution. In an extreme mismatch scenario, we find that without ECN, PIAS's performance suffers with medium flows and is worse than DCTCP. However, with ECN, PIAS effectively mitigates this problem, and is better than, or at least comparable to DCTCP.
- **What is the effect of the optimal demotion thresholds?** Compared to PIAS with thresholds derived from simple heuristics, PIAS with the optimal demotion thresholds achieves up for $\sim$8.3% improvement in medium flows, which improves the overall performance.

Impact of number of queues: In general, the more queues we use, the better we can segregate different flows, thus improving



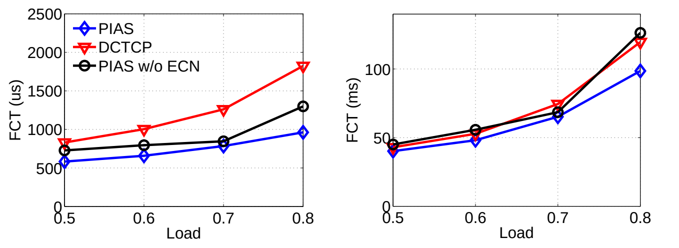
(a) (0,100KB]: Avg (b) (100KB,10MB]: Avg

Fig. 12: Web search workload with different numbers of queues



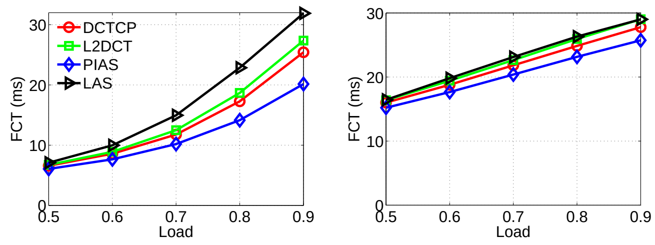
(a) (0,100KB]: Avg (b) (100KB,10MB]: Avg

Fig. 14: Web search workload with different thresholds



(a) (0,100KB]: Avg (b) (100KB,10MB]: Avg

Fig. 13: Web search workload with mismatch thresholds derived from data mining workload



(a) Web search workload (b) Data mining workload

Fig. 15: Overall average flow completion time

overall performance. For this experiment, we generate traffic using the web search workload and do the evaluation with 2, 4 and 8 priority queues. The results are shown in Figure 12. We observe that three schemes achieve the similar average FCT for short flows. As expected, the average FCT of medium flows improves with the increasing number of queues. For example, at 80% load, PIAS with 4 queues and 8 queues provide similar performance, but improve the FCT by 14.5% compared to 2 queues. The takeaway is that PIAS can effectively reduce FCT of short flows even with 2 queues and more queues can further improve PIAS's overall performance.

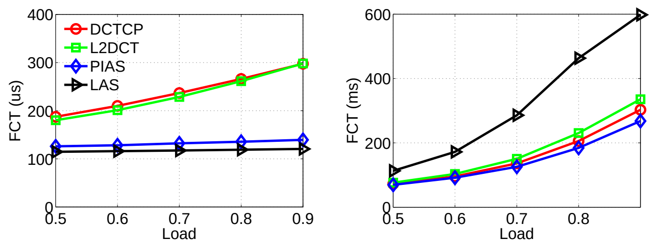
Effect of ECN under thresholds–traffic mismatch: We evaluate the performance of the web search workload while using the optimal demotion thresholds derived from the data mining workload. We compare PIAS, PIAS without ECN, and DCTCP. Figure 13 shows the results of the average FCT of short and medium flows. Both PIAS and PIAS without ECN greatly outperforms DCTCP in short flows. PIAS is slightly better than PIAS without ECN since it can efficiently reduces the switch buffer occupancy when many small flows collide in the highest priority queue. Furthermore, PIAS without ECN generally shows the worst performance for medium flows. This is because, due to the mismatch between demoting thresholds and traffic distribution, medium flows and large flows coexist in lower priority queues. Without ECN, packets from medium flows would experience queueing delay behind large flows (or get dropped). With ECN, PIAS effectively mitigates this side-effect by keeping low buffer occupancy as explained in §$III-B.2$.

Impact of demotion thresholds: To explore the effectiveness of our optimal demotion threshold setting, we compare the optimized PIAS with the PIAS using thresholds derived from the equal split heuristic as [13]. More specifically, given a flow size distribution and N queues, we set the first threshold to the size of $100/N$th percentile flow, the second threshold to the size of $200/N$th percentile flow, and so on. We run the web search workload at 80% load and summarize results in the Figure 14. We test PIAS with 2 and 4 queues. We observe that there is an obvious improvement in the average FCT of medium flows with the optimized thresholds. Specifically, PIAS (4-queue) with the optimized thresholds can achieve $\sim$8.3% lower FCT for medium flows than that of equal split, and a $\sim$5.8% improvement for the 2-queue PIAS. This partially validates the effectiveness of our optimal threshold setting.

C. Large-scale NS-2 Simulations

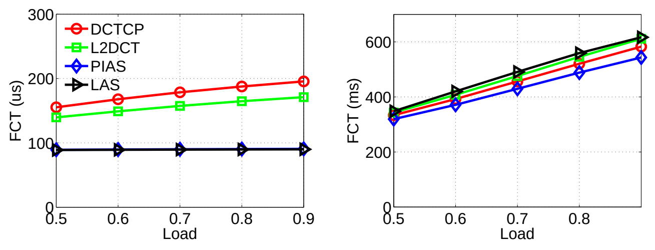
We use ns-2 [9] simulations to answer four questions.

- **How does PIAS perform compared to information-agnostic schemes?** PIAS outperforms DCTCP [11] and L2DCT [34] in general, and significantly improves their average FCTs for short flows by up to 50%. Furthermore, PIAS is close to LAS for short flows and greatly outperforms LAS for long flows, reducing its average FCT by up to 55% in the web search workload.
- **How does PIAS perform compared to information-aware schemes?** As a practical information-agonistic scheme, PIAS can also deliver comparable performance to a clean-slate information-aware design, pFabric [13], in certain scenarios. For example, it only has a 1.1% gap to pFabric for short flows in the data mining workload.
- **How does PIAS perform in the oversubscribed network?** In a 3:1 oversubscribed network with ECMP load



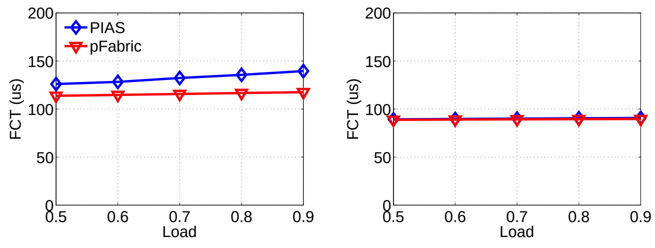
(a) (0,100KB]: Avg (b) (10MB,∞): Avg

Fig. 16: Web search workload: FCT



(a) (0,100KB]: Avg (b) (10MB,∞): Avg

Fig. 17: Data mining workload: FCT



(a) Web search workload (b) Data mining workload

Fig. 18: Average FCT for (0,100KB]

balancing, PIAS still delivers very good performance. Compared to DCTCP, the average FCT for the short flows with PIAS is up to ∼57% lower in the web search workload.

- **How does PIAS perform in the heterogeneous workload?** We create a heterogeneous traffic pattern where different links have different flow size distributions. In such challenging workload, PIAS still obviously outperforms DCTCP.

Setting: We use a leaf-spine topology with 9 leaf (ToR) switches to 4 spine (Core) switches. Each leaf switch has 16 10Gbps downlinks (144 hosts) and 4 40Gbps uplinks to the spine, forming a non-oversubscribed network. The base end-to-end round-trip time across the spine (4 hops) is 85.2μs. We use packet spraying [22] for load balancing and disable dupACKs to avoid packet reordering. The TCP initial window is 70 packets, which approximately equals to the bandwidth-delay product (BDP). The ECN marking threshold is 65 packets. For pFabric, each switch port has 210KB ($\sim 2\times BDP$) buffer. For the other schemes, each switch port has 360KB buffer that is completely shared by all the queues. The TCP RTOmin is 250μs ($\sim 3\times RTT$) for pFabric and 2ms for the others. We run 100000 flows for each simulation. Again, we use the web search and data mining workloads as above.

1) Comparison with Information-agnostic Schemes: We mainly compare PIAS with three other information-agnostic schemes: DCTCP, L2DCT [34] and LAS [39] (pFabric using bytes sent as the priority).

Overall performance: Figure 15 shows the average FCT of information-agnostic schemes under different workloads and load levels. From the figure, we see that PIAS delivers the best overall performance. By contrast, the performance of LAS is varied. PIAS significantly outperforms LAS by 37% (at 90% load) in the web search workload. This is because PIAS effectively mitigates the starvation between long flows unlike LAS. In the data mining workload, there are not so many large flows on the same link concurrently. As a result, LAS does not suffer from starvation as significantly.

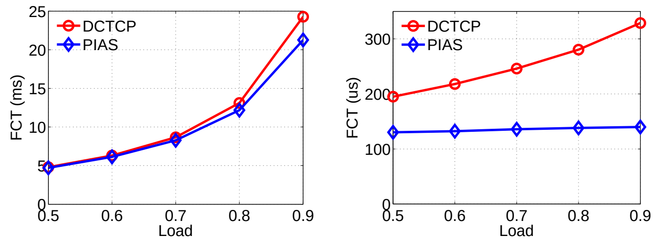
Breakdown by flow size: We now breakdown the average FCT across different flow sizes: (0, 100KB] and (10MB, ∞) (Figure 16 and 17).

For short flows in (0,100KB], we find that PIAS significantly outperforms both DCTCP and L2DCT, improving the average FCT by up to ∼50%. This is because DCTCP and L2DCT use reactive rate control at the end hosts, which is not as effective as PIAS for in-network flow scheduling. We further observe that PIAS achieves similar performance as LAS for short flows. PIAS only performs slightly worse than LAS in the web search workload when some packets from short flows get dropped.

For long flows in (10MB,∞), we find that PIAS performs significantly better than LAS in the web search workload (55% reduction in FCT at 90% load). This is because, in the web search workload, it is common that multiple large flows are present in the same link. In such scenarios, LAS always stops older flows to send new flows. Since large flows usually take a very long time to complete, it causes a serious starvation problem. However, with PIAS, large flows receive their fair share in the lowest priority queue, which mitigates this problem. Furthermore, PIAS slightly outperforms DCTCP and L2DCT for both workloads. We note that L2DCT is marginally worse than DCTCP for large flows. We attribute this into L2DCT's slower increasing rate in the congestion avoidance phase.

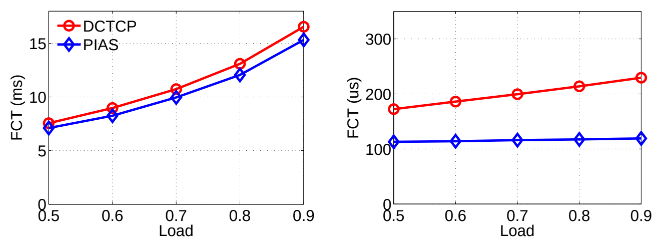
2) Comparison with Ideal Information-aware Schemes: We compare PIAS to an ideal information-aware approach for DCN transport, pFabric [13], on small flows of the two workloads. We note that the most recent work PASE [33] can achieve better performance than pFabric in particular scenarios (*e.g.*, very high load and single rack). However in our topology setting with realistic workloads, pFabric is better than PASE and PDQ [27], and achieves near-optimal performance. Thus, we directly compare PIAS with pFabric.

The result is shown in Figure 18. In general, PIAS delivers comparable average FCT for short flows as pFabric, particularly within 1.1% in the data mining workload. We find that the



(a) Overall: Avg (b) (0,100KB]: Avg

Fig. 19: Web search workload in a 3:1 oversubscribed network with ECMP load balancing.



(a) Overall: Avg (b) (0,100KB]: Avg

Fig. 20: Heterogeneous traffic pattern: FCT

gap between PIAS and pFabric is smaller in the data mining workload than that in the web search workload. This is mainly due to the fact that the data mining workload is more skewed than the web search workload. Around 82% flows in data mining are smaller than 100KB, while only 54% of flows in web search are smaller than 100KB. For the web search workload, it is more likely that large flows coexist with short flows in the high priority queues temporarily, increasing the queueing delay for short flows. pFabric, by assuming prior knowledge of flow sizes, is immune to such problem.

3) Performance in oversubscribed network: We evaluate PIAS in a 3:1 oversubscribed network with ECMP load balancing. In this network, there are 12 leaf switches and 4 spine switches. Each leaf switch is connected to 12 hosts and 4 spine switches with 10Gbps links. Given that the source and destination of each flow is generated randomly, $\frac{11}{12}$ of traffic is intra-ToR and the rest is inter-ToR traffic. Hence, the load at the fabric's core is $\frac{11}{12}\times3{=}2.75$ the load at the edge. We repeat the web search workload and compare PIAS with DCTCP. Figure 19 gives the results. Note that the load in the figure is at the fabric's core. Compared to DCTCP, PIAS achieves up to $\sim$57% and $\sim$12% lower average FCT for short flows and all the flows, respectively.

4) Performance in heterogeneous traffic: In previous simulations, traffic patterns are highly homogeneous since all the links have the same flow size distribution. In this simulation, we create a heterogeneous traffic pattern instead. As we have 144 hosts, we have 144×143 communication pairs in total. For a communication pair $(i,\ j)$[5], we run the web search workload if $i<j$. Otherwise, we run the data mining workload. Consequently, different links have different flow size distributions. Such heterogenous traffic is very challenging for PIAS as the demotion threshold mismatch unavoidably happens in everywhere. We evaluate the performance of PIAS (using demotion thresholds derived for the web search workload) and DCTCP using such heterogeneous workload. As Figure 20 shows, compared to DCTCP, PIAS achieves up to $\sim$48% and $\sim$8.1% lower average FCT for short flows and all the flows, respectively. This further shows the robustness of PIAS in large-scale production data center networks.

VI. Related Work

We classify prior work on minimizing FCT in DCNs into two categories: information-agnostic solutions (*e.g.*, [11, 12, 32, 34, 47]) and information-aware solutions (*e.g.*, [13, 27, 33]).

Information-agnostic solutions [11, 12, 32, 34, 47] generally improve the FCT for short flows by keeping low queue occupancy. For example, DCTCP [11] and DCQCN [47] try to keep the fabric queues small by employing ECN-based adaptive congestion control algorithms to throttle elephant flows. HULL [12] further improves the latency of DCTCP at the cost of trading network bandwidth. In summary, these solutions mainly perform end-host based rate control which is ineffective for flow scheduling. By contrast, PIAS leverages in-network priority queues to emulate SJF for flow scheduling, which is more efficient in terms of FCT minimization.

Information-aware solutions [13, 27, 33] attempt to approximate ideal Shortest Remaining Processing Time (SRPT) scheduling. For example, PDQ [27] employs switch arbitration and uses explicit rate control for flow scheduling. pFabric [13] decouples flow scheduling from rate control and achieves near-optimal FCT with decentralized in-network prioritization. PASE [33] synthesizes the strengths of previous solutions to provide good performance. In general, these solutions can potentially provide ideal performance, but they require non-trivial modifications on switch hardware or a complex control plane for arbitration. By contrast, PIAS does not touch the switch hardware or require any arbitration in the control plane, while still minimizing FCT.

There are also some other efforts [19, 41, 43] focusing on meeting flow deadlines. D3 [43] assigns rates to flows according to their sizes and deadlines explicitly, whereas D2TCP [41] and MCP [19] add deadline-awareness to ECN-based congestion window adjustment implicitly. They all require prior knowledge of flow information and do not directly minimize FCT, unlike PIAS.

To achieve diverse scheduling goals, some generic scheduling architectures [24, 36] have been proposed recently. In theory, we can also leverage these architectures to enforce MLFQ scheduling policy. In practice, it is challenging to deploy these architectures in large-scale production data centers. For example, Fastpass [36] uses a centralized scheduler, which suffers from failures and centralized scheduling overheads.

[5] i and j are host IDs.

pHost [24] requires the core of the underlaying network is congestion-free, which may not hold in practice [37]. By contrast, PIAS is easy to deploy in production data centers since it just leverages features of commodity switches without extra requirements.

VII. Conclusion

In this paper, we introduced PIAS, a solution that exploits existing commodity switches in DCNs to minimize the average FCT for flows, especially the smaller ones, without assuming any prior knowledge of flow sizes. We have implemented a PIAS prototype using all commodity hardware and evaluated PIAS through a series of small-scale testbed experiments as well as large-scale packet-level ns-2 simulations. Both our implementation and evaluation results demonstrate that PIAS is a viable solution that achieves all our design goals.

References

[1] http://www.pica8.com/documents/pica8-datasheet-picos.pdf.

[2] "Apache Storm," https://storm.incubator.apache.org/.

[3] "Cisco Nexus 5500 Series NX-OS Quality of Service Configuration Guide," http://www.cisco.com/c/en/us/td/docs/switches/datacenter/nexus5500/sw/qos/7x/b_5500_QoS_Config_7x.pdf.

[4] "Hadoop," http://hadoop.apache.org/.

[5] "Linux netfilter," http://www.netfilter.org.

[6] "Memcached," http://memcached.org/.

[7] "Microsoft SQL Server," http://www.microsoft.com/en-us/server-cloud/products/sql-server/.

[8] "OpenStack Object Storage," http://docs.openstack.org/api/openstack-object-storage/1.0/content/chunked-transfer-encoding.html.

[9] "The Network Simulator NS-2," http://www.isi.edu/nsnam/ns/.

[10] M. Alizadeh, T. Edsall, S. Dharmapurikar, R. Vaidyanathan, K. Chu, A. Fingerhut, V. T. Lam, F. Matus, R. Pan, N. Yadav, and G. Varghese, "CONGA: Distributed Congestion-aware Load Balancing for Datacenters," in *SIGCOMM 2014*.

[11] 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*.

[12] 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*.

[13] M. Alizadeh, S. Yang, M. Sharif, S. Katti, N. McKeown, B. Prabhakar, and S. Shenker, "pfabric: Minimal near-optimal datacenter transport," in *SIGCOMM 2013*.

[14] W. Bai, L. Chen, K. Chen, D. Han, C. Tian, and H. Wang, "Information-Agnostic Flow Scheduling for Commodity Data Centers," in *NSDI 2015*.

[15] H. Ballani, P. Costa, C. Gkantsidis, M. P. Grosvenor, T. Karagiannis, L. Koromilas, and G. O'Shea, "Enabling End-Host Network Functions."

[16] T. Benson, A. Akella, and D. A. Maltz, "Network Traffic Characteristics of Data Centers in the Wild," in *IMC 2010*.

[17] T. Benson, A. Anand, A. Akella, and M. Zhang, "Understanding data center traffic characteristics," *ACM SIGCOMM Computer Communication Review*, vol. 40, no. 1, pp. 92–99, 2010.

[18] J. Cao, R. Xia, P. Yang, C. Guo, G. Lu, L. Yuan, Y. Zheng, H. Wu, Y. Xiong, and D. Maltz, "Per-packet load-balanced, low-latency routing for clos-based data center networks," in *CoNEXT 2013*.

[19] L. Chen, S. Hu, K. Chen, H. Wu, and D. Tsang, "Towards Minimal-Delay Deadline-Driven Data Center TCP," in *HotNets 2013*.

[20] A. Chipperfield, P. Fleming, H. Pohlheim, and C. Fonseca, "Genetic algorithm toolbox for use with MATLAB," 1994.

[21] A. Demers, S. Keshav, and S. Shenker, "Analysis and Simulation of a Fair Queueing Algorithm," in *SIGCOMM 1989*.

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

[23] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, "Hypertext transfer protocol–HTTP/1.1, 1999," *RFC2616*, 2006.

[24] 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*.

[25] 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*.

[26] K. He, E. Rozner, K. Agarwal, W. Felter, J. Carter, and A. Akella, "Presto: Edge-based Load Balancing for Fast Datacenter Networks," in *SIGCOMM 2015*.

[27] C.-Y. Hong, M. Caesar, and P. Godfrey, "Finishing flows quickly with preemptive scheduling," in *SIGCOMM 2012*.

[28] K. Jang, J. Sherry, H. Ballani, and T. Moncaster, "Silo: Predictable Message Completion Time in the Cloud," Tech. Rep. MSR-TR-2013-95, 2013.

[29] V. Jeyakumar, M. Alizadeh, D. Mazieres, B. Prabhakar, C. Kim, and A. Greenberg, "EyeQ: practical network performance isolation at the edge," in *NSDI 2013*.

[30] B. Kalyanasundaram and K. R. Pruhs, "Minimizing flow time nonclairvoyantly," *Journal of the ACM (JACM)*, vol. 50, no. 4, pp. 551–567, 2003.

[31] S. Kandula, S. Sengupta, A. Greenberg, P. Patel, and R. Chaiken, "The nature of data center traffic: measurements & analysis," in *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference*. ACM, 2009, pp. 202–208.

[32] R. Mittal, V. T. Lam, N. Dukkipati, E. Blem, H. Wassel, M. Ghobadi, A. Vahdat, Y. Wang, D. Wetherall, and D. Zats, "TIMELY: RTT-based Congestion Control for the Datacenter," in *SIGCOMM 2015*.

[33] A. Munir, G. Baig, S. M. Irteza, I. A. Qazi, A. Liu, and F. Dogar, "Friends, not Foes - Synthesizing Existing Transport Strategies for Data Center Networks," in *SIGCOMM 2014*.

[34] 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 2013*.

[35] Y. Peng, K. Chen, G. Wang, W. Bai, Z. Ma, and L. Gu, "HadoopWatch: A First Step Towards Comprehensive Traffic Forecasting in Cloud Computing," in *INFOCOM 2014*.

[36] J. Perry, A. Ousterhout, H. Balakrishnan, D. Shah, and H. Fugal, "Fastpass: A Centralized Zero-Queue Datacenter Network," in *SIGCOMM 2014*.

[37] L. Popa, P. Yalagandula, S. Banerjee, J. C. Mogul, Y. Turner, and J. R. Santos, "ElasticSwitch: Practical Work-conserving Bandwidth Guarantees for Cloud Computing," in *SIGCOMM 2013*.

[38] S. Radhakrishnan, Y. Geng, V. Jeyakumar, A. Kabbani, G. Porter, and A. Vahdat, "SENIC: scalable NIC for end-host rate limiting," in *NSDI 2014*.

[39] I. A. Rai, G. Urvoy-Keller, M. K. Vernon, and E. W. Biersack, "Performance Analysis of LAS-based Scheduling Disciplines in a Packet Switched Network," in *SIGMETRICS 2004*.

[40] K. Ramakrishnan, S. Floyd, and D. Black, "RFC 3168: The addition of explicit congestion notification (ECN) to IP," 2001.

[41] B. Vamanan, J. Hasan, and T. Vijaykumar, "Deadline-aware datacenter tcp (d2tcp)," in *SIGCOMM 2012*.

[42] V. Vasudevan, A. Phanishayee, H. Shah, E. Krevat, D. G. Andersen, G. R. Ganger, G. A. Gibson, and B. Mueller, "Safe and effective fine-grained TCP retransmissions for datacenter communication," in *SIGCOMM 2009*.

[43] C. Wilson, H. Ballani, T. Karagiannis, and A. Rowtron, "Better never than late: Meeting deadlines in datacenter networks," in *SIGCOMM 2011*.

[44] H. Wu, J. Ju, G. Lu, C. Guo, Y. Xiong, and Y. Zhang, "Tuning ECN for data center networks," in *CoNEXT 2012*.

[45] Y. Xu, M. Bailey, B. Noble, and F. Jahanian, "Small is Better: Avoiding Latency Traps in Virtualized Data Centers," in *SOCC 2013*.

[46] M. Yu, A. G. Greenberg, D. A. Maltz, J. Rexford, L. Yuan, S. Kandula, and C. Kim, "Profiling Network Performance for Multi-tier Data Center Applications." in *NSDI 2011*.

[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*.