Datacenter Congestion Control: Identifying what is essential and making it practical

Aisha Mushtaq
UC Berkeley
aisha@cs.berkeley.edu

Radhika Mittal
UC Berkeley
radhika@cs.berkeley.edu

James McCauley
UC Berkeley, ICSI
jmccauley@cs.berkeley.edu

Mohammad Alizadeh
MIT
alizadeh@csail.mit.edu

Sylvia Ratnasamy
UC Berkeley
sylvia@cs.berkeley.edu

Scott Shenker
UC Berkeley, ICSI
shenker@icsi.berkeley.edu

ABSTRACT

Recent years have seen a slew of papers on datacenter congestion control mechanisms. In this position paper, we ask whether the bulk of this research is needed for the most common case where the performance goal is reducing average Flow Completion Time. We raise this question because we find that, out of all the possible variations one could make in congestion control algorithms, the most essential feature is the switch scheduling algorithm. More specifically, we find that congestion control mechanisms that use Shortest-Remaining-Processing-Time (SRPT) achieve superior performance as long as the rate-setting algorithm at the end host is reasonable. We further find that while SRPT's performance is quite robust to end host behaviors, the performance of schemes that use scheduling algorithms like FIFO and Fair Queuing depend far more crucially on the rate-setting algorithm, and their performance is typically worse than what can be achieved with SRPT. Given these findings, we then ask whether it is practical to realize SRPT in switches without requiring custom hardware. We observe that *approximate and deployable SRPT* (ADS) designs exist, which leverage the small number of priority queues supported in almost all commodity switches, and require only software changes in the host and, perhaps, the switches. Our evaluations with one very simple ADS design shows that it can achieve performance close to true SRPT and is significantly better than FIFO. Thus, the answer to our basic question - whether the bulk of recent research on datacenter congestion control algorithms is needed for the common case - is no.

1. INTRODUCTION

In its never-ending quest to improve datacenter performance, the research community has produced a continuing stream of papers on datacenter congestion control. These proposals differ along many dimensions, including how congestion is detected and signaled, how end hosts adapt to congestion, and how packets are scheduled by the switches. The result is a cornucopia of datacenter congestion control schemes embodying a wide variety of designs and displaying varying levels of performance. While these efforts have had significant practical impact, they have not provided a systematic understanding of the congestion control design space. As a result, after all these years of promising individual proposals – each eloquently described and thoroughly evaluated – we as a community do not have a broad overarching understanding of what factors are most important in achieving good congestion control performance in datacenters.

In this paper, we take a first step towards filling this void with an initial factor analysis of datacenter congestion control schemes. Our goal is not to invent new congestion control schemes, but to understand which factors are most important for good performance. We focus on scenarios where minimizing some average measure of flow completion times (FCT) is the performance goal (as opposed to more sophisticated resource allocation goals [15, 19, 11] or meeting explicit deadlines [9, 21, 22]) and ask two questions:

1. What factors (i.e., which particular design decisions) are most essential to achieving good performance?
2. Can we deploy such designs easily?

To answer the first question, we decompose datacenter congestion control schemes into two components: (i) *scheduling* (the order in which packets are forwarded by network switches) and (ii) *rate-setting* (how hosts respond to congestion and how that congestion is signaled by the switches).[1] Our factor analysis involves looking at a variety of designs for each component. For scheduling algorithms, we look at FIFO, Fair Queueing (FQ), Shortest-Job-First (SJF), and Shortest-Remaining-Processing-Time (SRPT). For rate-setting, we primarily look at TCP (which uses packet drops to signal congestion), DCTCP (which uses ECN to signal congestion), and minTCP (a minimal form of TCP introduced in [2]).[2]

Our findings on this first question (Section 2) are quite clear. The most important factor in achieving good performance is using SRPT (or its close cousin, SJF) for packet scheduling. Our results can be summarized in two statements: (1) A rate-setting scheme achieves its best performance when used with SRPT, and (2) with SRPT, a wide range of rate-setting schemes achieve near-optimal performance.

Note that SRPT is neither necessary nor sufficient for good performance. Clearly, when combined with obviously inappropriate rate-setting schemes (*e.g.,* not increasing the window size sufficiently quickly when flows start, or having extremely long timeout values), the resulting performance with SRPT can be significantly suboptimal; thus, SRPT is not sufficient. In addition, there are several congestion control proposals that mimic the behavior of

[1] Note that this decomposition does not include designs such as pHost [8] where the endpoints engage in sophisticated interactions with each other to determine when to send packets. Our focus is on schemes where the ends adjust their rate in response to simple congestion signals from the network.

[2] While we do not have the space to show them, we also have results on PCC [6] and RCP [7].

SRPT with simple FIFO queues. For instance, pHost [8] approximates SRPT through complicated host negotiations, and achieves near-optimal performance (but requires very careful tuning); thus, SRPT is not necessary. However, our point is that with SRPT, near-optimal performance is easy to achieve.

The second question arises because the SRPT scheduling algorithm is not available on commodity switches, and is not even supported by newer scheduling mechanisms like PIFO [20]. Moreover, the remaining-bytes information needed for SRPT is not available in today's transport protocols. Thus, it is not clear whether SRPT-inspired designs are easily deployable. However, our answer to this second question is a resounding "yes". As discussed in Section 3, one can easily modify end hosts to supply an approximation to the remaining-bytes information, and one can leverage the priority-scheduled FIFO queues present on most commodity switches to roughly emulate SRPT scheduling. Thus, we claim that *approximate and deployable SRPT* designs exist (we call them ADS designs), which only require simple software changes in the host, and perhaps in switches (depending on the exact choice of ADS mechanism).

The idea of using SRPT is hardly new, as it was introduced in pFabric [2], and the notion that one can use a fixed number of standard priority-scheduled queues to emulate SRPT was discussed in [2, 3, 8, 14, 12]. Our goal in writing this paper was twofold. First, it was to observe that (based on our factor analysis) SRPT is the one essential aspect of pFabric's design, and everything else could be modified in various ways while maintaining near-optimal performance. Second, and more importantly, our goal was to argue that for minimizing some average measure of FCT in datacenters, we essentially have all the mechanisms we need. That is, we now know that SRPT is *the* crucial factor in achieving good performance, and that we now have ways of deploying reasonable approximations to SRPT without requiring hardware changes to routers. While more work is needed to optimize such techniques (see Section 4), these ADS designs are clearly better than the status quo. Thus, our message to the commercial community is simple: *in order to minimize average measures of FCT in datacenters, we should be rapidly moving towards adoption of the ADS approach.* Our message to the research community is even simpler: *if you are interested in minimizing average measures of FCT in datacenters, look no further than ADS.*

2. FACTOR ANALYSIS

2.1 General Methodology

In our investigation, we focus on the most widely accepted measure of performance: minimizing flow completion times (FCT). However, rather than focusing primarily on the average FCT (which is often dominated by the completion time of the largest flows), we use the average *slowdown* (comparing the flow completion time of each flow to how long it would have taken to complete if it were sent at line-rate [2, 3, 8]) as our basic measure of performance because it captures the behavior across a spectrum of flow sizes. However, we also report the average FCT and tail FCT.

We consider the following rate-setting schemes for hosts and scheduling algorithms for switches, as listed below.

Rate-Setting Schemes:

(i) minTCP: This is the rate control scheme used in pFabric [2], which starts sending at line rate and uses only timeouts to detect congestion.

(ii) TCP: We use TCP SACK, which employs duplicate ACKs in addition to timeouts to detect losses, and uses AIMD for rate control.

(iii) DCTCP: It has the same rate control as TCP but additionally uses the fraction of ECN marks to signal congestion.

Scheduling Algorithms:

(i) FIFO: First In First Out scheduling.

(ii) FQ: Fair Queuing as described in [5].

(iii) SJF: Shortest Job First. Flows are prioritized based on their absolute size (shorter flows get higher priority[3]).

(iv) SRPT: Shortest Remaining Processing Time, as used in pFabric [2]. Flows with smaller remaining time (fewer outstanding bytes) get higher priority.[3] The switches also implement a *starvation prevention* mechanism where the earliest arriving packet of the flow containing the highest priority packet is selected for transmission.

2.2 Experimental Setup

Topology. We use the same spine-leaf topology as in [2], which interconnects 144 hosts through 9 leaf switches connected to 4 spine switches in a complete bipartite graph. It is a full bisection bandwidth fabric, with each leaf switch having sixteen 10 Gbps downlinks (to the hosts) and four 40 Gbps uplinks (to the spine). The bandwidth delay product (BDP) is roughly 14 packets (with each packet carrying 1500 bytes). The end-to-end round-trip time (RTT) across the spine is 16 μs. We set the buffer sizes at the switches and the end hosts to 500 KB (shared among queues, if any). We use flow-based ECMP [10] to balance load.

Workload. We consider loads where flows arrive according to a Poisson process and are sent from a random source to a random destination server. For our default scenario we use a heavy-tailed distribution [4] where 50% of the flows are of 1 KB (1 packet), 35% are between (1 KB, 200 KB] and 15% are between (200 KB, 3 MB]. The inter-arrival time between flows is computed such that the average link utilization is 70%. To test the robustness of our results, we also try other workloads and utilization levels as discussed in §2.3.2.

Default parameters. We set the initial window sizes to the bandwidth-delay-product (BDP) packets (14 packets for our topology), and the minimum timeout is set to three times the two-way network propagation delay ($3 \times RTT$). The ECN threshold for DCTCP is set to 15 packets.

Simulation Methodology. We run simulations using the ns-2 [16] simulator. For our default scenario, we run the simulations for a total of 1.5 s of simulated time, recording the FCTs of all flows that started between 0.05 s and 0.15 s. This results in a total of roughly 44000 flows being considered for our default scenario.

2.3 Results

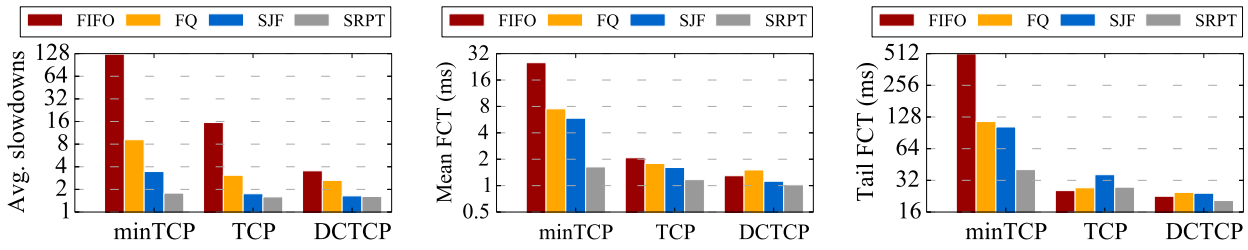
2.3.1 Results on Default Scenario

Average Slowdown. Figure 1(a) shows the slowdowns for our default scenario, with every combination of scheduling algorithm (from the set {FIFO, FQ, SJF, SRPT}) and rate-setting scheme (from the set {minTCP, TCP, DCTCP}). The following are the key takeaways.

(1) Comparing performance across scheduling algorithms: Overall, SRPT performs much better in terms of slowdown than FIFO or FQ. For example, TCP performs 9.7× better with SRPT than with FIFO and 1.8× better with SRPT than with FQ. Although the relative difference between FIFO and SRPT is smaller for the more sophisticated DCTCP scheme, it is still as high as 2.2×. minTCP (which is by far the least sophisticated) varies by 70× across the scheduling schemes.

(2) Comparing performance across rate-setting schemes:
SRPT is extremely *robust*, in that the rate-setting scheme typically

[3] ACKs are assigned the highest priority, as in [2].



(a) Average Slowdown (b) Average FCT (c) 99%ile FCT

Figure 1: Results for our default scenario (a heavy tailed distribution running on a fat-tree topology at 70% utilization)

has only minimal impact on performance when using SRPT scheduling; the slowdowns of various rate-setting schemes are within 1%-5% of each other with SRPT. In contrast, the performance with FIFO scheduling is highly sensitive to the rate-setting scheme.

(3) SRPT vs SJF: The performance of SJF is very close to SRPT for rate-setting schemes that back off reasonably in the face of congestion, such as TCP and DCTCP. The slowdown of TCP with SJF is within 5% of TCP with SRPT scheduling, while that of DCTCP is within 1.5% of DCTCP with SRPT. However, the difference between SRPT and SJF with minTCP is significantly higher. This is because minTCP relies only on timeouts to detect congestion and back off. Hence, long flows (which, in the absence of SRPT's priority escalation mechanism, have the lowest absolute priority with SJF) do not back off soon enough at the onset of congestion, resulting in increased packet losses and repeated timeouts.[4]

Average FCT. Figure 1(b) shows the average FCTs obtained for the default scenario. The general conclusions that we drew for average slowdowns above hold for average FCTs as well.

Tail FCT. For completeness, we also looked at the tail FCT (which is dominated by the behavior of long flows). Given that SRPT/SJF prioritize shorter flows over the long flows, one would expect these scheduling schemes to have a huge negative impact on the tail behavior. Very surprisingly, for most cases, SRPT performance is better than other scheduling mechanisms even for the tails of the distributions. Figure 1(c) shows the tail (99^{th} percentile) FCTs for our default scenario. SRPT continues to perform the best when used with DCTCP and minTCP. Though the best performer for tail FCT with TCP is FIFO, SRPT performance stays within 8% of it. However, the tail FCT in the datacenter is somewhat sensitive to rate-setting scheme even with SRPT (for example, note the difference between TCP and DCTCP). So while SRPT is not as robust with respect to tail performance as it is for the average performance, it performs reasonably well.[5]

2.3.2 *Evaluating Robustness of General Results*

We tested the robustness of our high level trends by varying several aspects of our experimental scenario.

Workload: We considered a variety of workloads, including the Facebook workload from [18], the web search workload from [1], and some non-conventional workloads: a uniform distribution where flow sizes are picked with equal probability from a set of fixed values in [1 KB, 3 MB], a bimodal distribution where 50% of the flows are of size 2 KB and 50% of the flows are of size 2 MB, and a point distribution where all flows are of size 2 MB.

Utilization: We varied the link utilization levels from our default value of 70% to lower values of 30% and 50%.

Varying Parameters: We repeated the experiments for our default scenario by varying the initial congestion window between 2 to 21 packets (our default value being $BDP = 14$ packets). We also varied the minimum timeout from 16μs to 1ms (RTT being 16μs).

For all these cases, our basic results remain unchanged, in that rate-setting schemes performed best with SRPT, and any "reasonable" rate-setting scheme resulted in near-optimal performance when used with SRPT. However, when the rate-setting scheme was unreasonable (e.g., extremely high timeout value), the results were far from optimal.

3. APPROXIMATE & DEPLOYABLE SRPT

While the previous section makes clear that SRPT is essential, it has what would seem to be a fatal flaw: no current switches are capable of supporting SRPT at line rate. Fortunately, there are approximations of SRPT which are amenable to implementation in commodity hardware that perform almost as well as true SRPT. In the following subsection, we discuss the ADS design space, and touch on the various related work in it. Following that, we focus on our own contribution to this space which assumes that the software (but not the hardware) of commodity switches can be changed. We end the section by showing results from this design.

3.1 ADS Design Space

As a reminder, in true SRPT, the switch must transmit the oldest packet from the flow with the fewest remaining bytes. Assuming that the remaining-bytes information is contained in each packet, the switch must both (i) do priority scheduling at arbitrarily fine granularity, and (ii) pick the oldest packet from the flow whose packet is at the head of the queue (to prevent starvation of old packets). Commodity switches cannot support either of these necessary behaviors.

Acknowledging this, pFabric [2] contained a preliminary effort to approximate this behavior using a feature that many commodity switches *do* support – a small number (often eight) of independent priority-scheduled queues, where all packets from a higher priority queue are sent before packets from lower priority queues. Packets with similar numbers of remaining bytes are binned together into the available queues using some heuristic, a process we call *priority mapping*.[6] To clarify our terminology, we assume that the field that goes into the packet is called its "priority", and this priority is used (by the priority mapping) to select the appropriate queue for that packet.

As one moves beyond the preliminary ADS work done in [2],

[4]This is in contrast to what is reported in [2], but there are differences in our default simulation settings, such as no packet-spraying and large packet buffers, that might explain the discrepancy.

[5]We should note that tail FCT is a questionable measure for these workloads, as it only reflects the treatment of the few very longest flows. Thus, we do not view the desirability of SRPT being impacted by its somewhat imperfect performance here.

[6]Note that in this approach one need not implement the starvation-prevention explicitly.

however, it becomes clear that there is a large design space of possible approximations, the dimensions of which we explore below.

Estimation of Flow Lengths. SRPT requires information about flow lengths in order to determine priority. However, this information does not exist in typical transport protocols (and is, in fact, somewhat counter to the "stream" service model provided by, *e.g.,* TCP), and so must be introduced in some way. pFabric suggests the addition of a new socket option whereby a modified application can provide a size hint to the network stack, which then inserts this hint into the packet header. PIAS [3] suggests an alternate technique which does not require modifying applications and works in cases where a flow size is not known a priori: essentially, flows are assumed short until proven long (*i.e.,* priorities start high and shift lower as the flow goes on). In the next subsection, we describe yet another approach.

How the priority mappings are computed. Somehow, a mapping must be made from a large priority space (*i.e.,* the space of all possible flow sizes) to a small number of queues. [2] and [3] present their own analytical approaches. [14] leverages hierarchy in the topology by employing a hierarchical set of algorithmic arbitrators distributed through the network which attempt to put a single flow (*e.g.,* the shortest one) in the high priority queues, with other queues holding packets from successively lower priority flows. In contrast, [8] computes priorities entirely at endhosts. In §3.2, we examine an extremely simple heuristic to compute priority mapping at each switch.

Traffic measurement. No matter the exact details of how the thresholds are computed, doing so requires some sort of measurement of the flow size distribution (and, for example, both [2] and [3] note that load is also important). This raises the issue of where these measurements are taken. [8], for example, takes measurements only at the hosts, and both [2] and [3] assume only global datacenter-wide measurement while acknowledging that this is suboptimal, as traffic may not be uniform across the topology. In the next subsection, we describe a practical scheme in which measurements can be taken at each switch.

Measurement/update frequency. If the computation is expensive or centralized, this may limit the update frequency. PIAS, for example, acknowledges the utility of periodic updating, but specifically leaves the issue to future work. The scheme we describe in the next subsection can do updates dynamically, and in our evaluation, we examine both dynamic updating and an extreme example of non dynamic updating.

Per-switch or network-wide thresholds. Is the priority mapping applied globally, or per-switch? [8], [3], and the preliminary work in [2] all examine the case where all switches process a given packet at the same priority. In the scheme we describe in §3.2, each switch does its own measurement and then computes independently based on its local measurements, so each switch may arrive at different thresholds if the traffic is not uniform.

How many queues are used. While many switches have eight queues, not all may be available for ADS. [12] in particular focuses on this issue by making good performance with only two queues an essential part of their design. We later evaluate our own scheme with both two and eight queues.

3.2 Our Approach

The overarching message of this paper is that we believe that many points in the ADS design space are very promising; indeed, the related work serves as indications that this is indeed the case. Here, we contribute to this gathering set of evidence by choosing some additional points in the space, fleshing out a practical design, and providing some preliminary evaluation. We make the following assumptions in the name of practicality:

No application modifications. This primarily affects how flow sizes are estimated, as it rules out the technique where applications provide flow size hints to the networking stack. To do this, we track the number of bytes the application has written to a socket which are still outstanding at the sender.[7] We use this as an estimate of the remaining flow size, capping it at 2MB and configuring the network stack to have write buffers of at least 3MB. By providing at least an extra 1MB of "slack space", typical applications which simply write as much and as quickly as the OS allows will stay above the 2MB threshold and therefore receive the minimum priority (all such flows are simply treated as "long"). Byte-level granularity is not necessary, as most packets will be sized to fit the MTU, so we divide the number of outstanding bytes by the MTU.

Commodity switching hardware. We attempt to target commodity switching hardware. We assume that switches have tables that can match on common fields, and that two operations can be performed when a packet matches a table entry: (i) a packet counter associated with the entry can be incremented, and (ii) the egress queue number for the packet can be read from the associated table entry. We further assume that the control plane software on such switches can be modified in order to query the counters and update the table entries and associated queue numbers. As with all the approximations we discuss in this section, we assume there are a number of individual FIFO queues where the scheduling between queues is strict priority. Finally, we must put the remaining flow size estimate (priority) into the packet somewhere. Again, to facilitate implementation on switching hardware without flexible matching hardware, we chose to co-opt the VID field of the 802.1Q VLAN header. This allows 12 bits, which is more than enough to reach our 2MB cap with 1500 byte packets.[8]

Within the switch, for each port, there are table entries containing all possible values for this field, and matching on this field loads the appropriate queue number. This table can be pre-populated statically if the workload distribution is known and is relatively stable. For this, we use the equal split heuristic originally discussed in [2] to map priorities, which attempts to equalize the number of packets in each queue.

We can use this same heuristic to dynamically populate this information in the switch. Thus, one can have "static" schemes, where the priority mapping is fixed (or changes on very slow time scales), or dynamic schemes where the priority mapping is updated quite rapidly (e.g., many times per second). For the dynamic scheme, the switch records the frequency of packet priorities it sees over a fixed time interval I. This gives us a sequence of distributions (i.e., a histogram) of the frequency of the packet priorities seen in an interval. After each time interval the resulting histogram is passed to the control plane of the switch to compute the priority mapping, and the priority counters are reset. The switch control plane uses a sliding window of the last W histograms to compute the priority mappings using the equal split heuristic described before.

Note that these table entries are the motivation for the 2MB cap on remaining flow size as well as the conversion from bytes-remaining to MTUs-remaining: doing so limits the required number of entries to about 1400 per port (2MB cap / 1500 byte MTU $\approx$ 1400 possible values). Granted, this is still a relatively large table: a 48 port switch would require about 67,200 entries. However, we believe this to be well within reach for commodity hardware, as many switches already support hundreds of thousands of exact-match entries.

[7] This could be used in bare-metal datacenters and container-based datacenters, but would require modifications to tenant software for use in VM-based datacenters.

[8] Depending on context, one may store this data elsewhere in the packet, for example, in a VXLAN [13] header as [12] advocates.

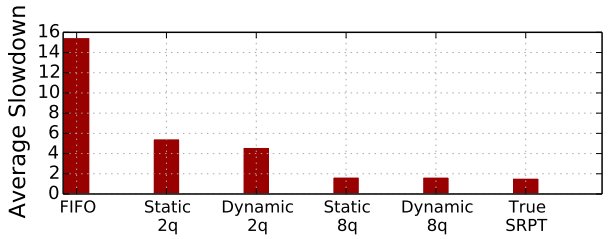


Figure 2: Average Slowdowns with SRPT emulation for our default heavy-tailed workload at 70% utilization.

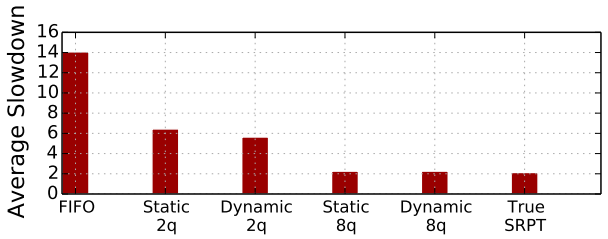


Figure 3: Average Slowdowns with SRPT emulation for uniform workload at 70% utilization.

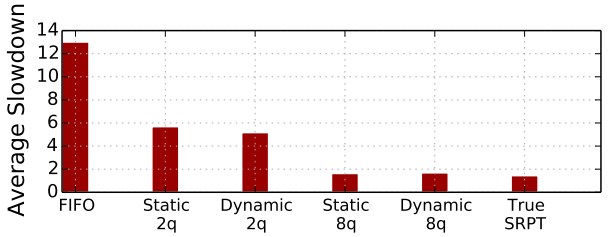


Figure 4: Average Slowdowns with SRPT emulation for the Facebook workload at 70% utilization.

In the following subsection, we evaluate an implementation of this design in the ns-2 network simulator [16], but first we wish to note that we also completed a preliminary "real world" implementation of both components. For the host portion, we implemented it as changes to the Linux TCP stack, comprised of the addition of about 20 LOC. As we do not have access to any proprietary switch SDKs, we targeted our switch prototype against the P4 bmv2 [17] software switch using 132 lines of P4 code. Although P4 is quite flexible, we note again that our design uses only fairly straightforward operations: it reads a value from a common field, does an exact match lookup in a table, increments a packet counter, and loads the queue number. All of these operations are also available in OpenFlow switches, and we could have easily chosen OpenFlow for our implementation. Moreover, we believe many switches support the required functionality, and – with appropriate access to their internals – could also implement this design.

3.3 Results

We now present our results for both static and dynamic approximation of SRPT as described in §3.2. We simulate the static scheme by recording the priority values seen by each switch when running *true* SRPT with the corresponding workload and then using the equal-split heuristic to compute the priority mapping. For the dynamic scheme, we set the update time interval I to be 5ms and the sliding window size W for the histograms to 10.

While most commodity switches typically have eight priority scheduled queues, not all of them may be available for use in a datacenter (for instance, the datacenter network operators might want to use different queues to differentiate between various traffic classes). We therefore present our results for both when we use all of the eight available queues and when we use only two queues for ADS. For comparison, we also show the corresponding results obtained with FIFO and with true SRPT as evaluated in §2.

3.3.1 Default Scenario

Figure 2 shows the average slowdowns obtained from our ADS approach. The first and the last bars show the performance with FIFO and true SRPT as baselines. The second and the third bars show the ADS results when only two queues are used with static and dynamic mapping respectively. The fourth and fifth bars shows the corresponding results with eight queues. The following are the key takeaways from these results:

(1) With eight queues, our ADS performs almost as well as true SRPT, with the average slowdowns being only 6.6% higher than true SRPT.

(2) ADS with two queues is not as good as that with eight queues, resulting in about 3× higher average slowdowns. In results not shown, with four queues the average slowdown is only 13% higher than that with eight queues.

(3) While there was little difference between the static and dynamic scheme for ADS with eight queues, the dynamic scheme performs better than the static scheme with two queues. We believe that this is because, unlike the static scheme, the dynamic scheme is able to adapt to the instantaneous changes in traffic pattern.

(4) ADS with even two queues performs significantly better than FIFO. The average slowdown obtained from dynamic scheme for ADS with two queues is more than 3× smaller than FIFO.

We would like to stress on this last point. While we might not be able to match the SRPT performance in a practical setting where only few priority queues are available, using even one extra queue gives significant improvement over FIFO.

3.3.2 Robustness

Varying Workload We evaluated our ADS schemes for the variety of workload distributions mentioned in §2.3.2. We present results for two of these. Figure 3 shows the ADS results for the uniform distribution where flow sizes are picked with equal probability from a set of fixed values in [1 KB, 3 MB]. Figure 4 shows the ADS result for the heavy-tailed distribution obtained from Facebook [18] with flow sizes ranging from 1KB to 100MB. We find that our main takeaways from §2.3.2 hold for these workloads (which have very different characterisitics) as well.

Using a Different Mapping for Static ADS: We also tested the impact of using a different workload distribution to compute static priority mappings for ADS. More precisely, we used the static mapping computed for our default heavy-tailed workload to approximate SRPT for other workloads. To our surprise, we found that for many realistic workloads (such as for Facebook and web search workloads), using a priority mapping from another workload performed almost as well as when the static mapping was computed from the actual workload. This indicates that using a fixed priority mapping computed from a heavy tailed workload divides the remaining flow size values across the fixed priority queues in a robust manner, such that the same mapping performs well across many different workloads. Note, however, that using a fixed static mapping might not work well across *all* workloads. For instance, one can come up with workloads where all of the flows would end up being mapped to the same priority queue, thus boiling down to FIFO.

Varying Parameters for Dynamic ADS: We tested the impact of varying the update interval I and the sliding window size W for the dynamic SRPT approximation. We varied the former from 0.5ms to 100ms and observed that it only affected the convergence time (the longer the update interval, the longer it takes for the switch to update the priority mappings), but had negligible impact on performance after convergence. We varied W between 1 and 50 and that too had little impact on the performance.

4. CONCLUSION

Our results lead us to conclude that there are a range of ADS designs that perform far better than FIFO, and in some cases reasonably close to true SRPT, for the goal of minimizing average FCT. Moreover, these designs could be deployed today. We urge the commercial community to begin testing these designs under operational conditions (because simulations can always miss some crucial aspect of reality) and move towards deployment.

For the research community, when one restricts attention to minimizing some average measure of FCT in datacenters, we see few major research issues left to tackle. However, there are some remaining open questions about ADS designs, which we list below.

- What are the limits of our ADS approach? That is, are there load patterns where our conclusions do not hold (such as loads with only small flows)? Are there scenarios (such as RDMA deployments, which involve PFCs) where our conclusions do not hold?
- Is there an adaptive scheme that does significantly better than our simple scheme?
- Are there any ADS designs that do significantly better (than our current designs) with 2 queues? We have not explored this aspect in any depth, so there is an opportunity for improvement.
- Are there other more complicated schemes that come even closer to true SRPT? Designs such as [8, 12] are attempts in this direction. However, even if successful, the additional complexity may not be warranted.
- To what extent do these results extend cleanly to other metrics, such as the maximal FCT rather than average?

5. REFERENCES

[1] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan. Data center tcp (dctcp). *ACM SIGCOMM computer communication review*, 41(4):63–74, 2011.

[2] M. Alizadeh, S. Yang, M. Sharif, S. Katti, N. McKeown, B. Prabhakar, and S. Shenker. pfabric: Minimal near-optimal datacenter transport. *ACM SIGCOMM Computer Communication Review*, 43(4):435–446, 2013.

[3] W. Bai, L. Chen, K. Chen, D. Han, C. Tian, and H. Wang. PIAS: information-agnostic flow scheduling for commodity data centers. In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*, pages 455–468, Oakland, CA, 2015. USENIX Association.

[4] T. Benson, A. Akella, and D. Maltz. Network Traffic Characteristics of Data Centers in the Wild. In *Proc. ACM Internet Measurement Conference (IMC)*, 2012.

[5] A. Demers, S. Keshav, and S. Shenker. Analysis and simulation of a fair queueing algorithm. In *ACM SIGCOMM Computer Communication Review*, volume 19, pages 1–12. ACM, 1989.

[6] M. Dong, Q. Li, D. Zarchy, P. B. Godfrey, and M. Schapira. Pcc: Re-architecting congestion control for consistent high performance. In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*, pages 395–408, 2015.

[7] N. Dukkipati and N. McKeown. Why flow-completion time is the right metric for congestion control. *ACM SIGCOMM Computer Communication Review*, 36(1):59–62, 2006.

[8] 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 *Proceedings of the CoNEXT*, 2015.

[9] C.-Y. Hong, M. Caesar, and P. Godfrey. Finishing flows quickly with preemptive scheduling. *ACM SIGCOMM Computer Communication Review*, 42(4):127–138, 2012.

[10] C. E. Hopps. Analysis of an equal-cost multi-path algorithm, 2000.

[11] A. Kabbani, M. Alizadeh, M. Yasuda, R. Pan, and B. Prabhakar. Af-qcn: Approximate fairness with quantized congestion notification for multi-tenanted data centers. In *Proceedings of the 2010 18th IEEE Symposium on High Performance Interconnects*, HOTI '10, pages 58–65, Washington, DC, USA, 2010. IEEE Computer Society.

[12] Y. Lu, G. Chen, L. Luo, K. Tan, Y. Xiong, X. Wang, and E. Chen. One more queue is enough: Minimizing flow completion time with explicit priority notification. In *INFOCOM, 2017 Proceedings IEEE*, 2017.

[13] M. Mahalingam, D. Dutt, K. Duda, P. Agarwal, L. Kreeger, T. Sridhar, M. Bursell, and C. Wright. Virtual eXtensible Local Area Network (VXLAN): A Framework for Overlaying Virtualized Layer 2 Networks over Layer 3 Networks. IETF RFC 7348 (Informational), 2014.

[14] A. Munir, G. Baig, S. M. Irteza, I. A. Qazi, A. X. Liu, and F. R. Dogar. Friends, not foes: synthesizing existing transport strategies for data center networks. In *ACM SIGCOMM Computer Communication Review*, volume 44, pages 491–502. ACM, 2014.

[15] K. Nagaraj, D. Bharadia, H. Mao, S. Chinchali, M. Alizadeh, and S. Katti. NUMFabric: Fast and Flexible Bandwidth Allocation in Datacenters. In *Proc. ACM SIGCOMM*, 2016.

[16] The network simulator - ns-2. http://www.isi.edu/nsnam/ns/.

[17] P4 behavioral model v2 (bmv2). http://github.com/p4lang/behavioral-model.

[18] A. Roy, H. Zeng, J. Bagga, G. Porter, and A. C. Snoeren. Inside the social network's (datacenter) network. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, SIGCOMM '15, pages 123–137, New York, NY, USA, 2015. ACM.

[19] A. Shieh, S. Kandula, A. Greenberg, C. Kim, and B. Saha. Sharing the data center network. In *Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation*, NSDI'11, pages 309–322, Berkeley, CA, USA, 2011. USENIX Association.

[20] A. Sivaraman, S. Subramanian, A. Agrawal, S. Chole, S.-T. Chuang, T. Edsall, M. Alizadeh, S. Katti, N. McKeown, and H. Balakrishnan. Towards programmable packet scheduling. In *Proceedings of the 14th ACM workshop on hot topics in networks*, page 23. ACM, 2015.

[21] B. Vamanan, J. Hasan, and T. Vijaykumar. Deadline-aware datacenter TCP (d2tcp). *ACM SIGCOMM Computer Communication Review*, 42(4):115–126, 2012.

[22] C. Wilson, H. Ballani, T. Karagiannis, and A. Rowtron. Better never than late: Meeting deadlines in datacenter networks. In *Proc. of ACM SIGCOMM*, 2011.