Size-based scheduling vs fairness for datacenter flows: a queuing perspective

James Roberts*
Independent researcher
France
james.walter.roberts@gmail.com

Dario Rossi
Huawei Paris Research Center
France
dario.rossi@huawei.com

ABSTRACT

Contrary to the conclusions of a recent body of work where approximate shortest remaining processing time first (SRPT) flow scheduling is advocated for datacenter networks, this paper aims to demonstrate that imposing fairness remains a preferable objective. We evaluate abstract queuing models by analysis and simulation to illustrate the non-optimality of SRPT under the reasonable assumptions that datacenter source-destination flows occur in *batches* and *bursts* and not, as usually assumed, individually at the instants of a Poisson process. Results for these models have significant implications for the design of bandwidth sharing strategies for datacenter networks. In particular, we propose a novel "virtual fair scheduling" algorithm that enforces fairness between batches and is arguably simple enough to be implemented in high speed devices.

1. INTRODUCTION

To realize low latency in datacenter networks (DCNs) it is now frequently proposed that flow scheduling should be size-based, referring to the well-known response time optimality of the shortest remaining processing time first (SRPT) policy [32]. Authors have devised practical algorithms that closely approximate SRPT and demonstrate significant latency reduction compared to max-min fairness, e.g., [5, 27, 28]. However, to demonstrate the advantages of SRPT, the cited papers adopt a simplified traffic model where source-destination flows of random size arrive according to a Poisson process. The objective of the present paper is to show that the claimed superiority does not hold under more realistic traffic models and fairness arguably remains a desirable scheduling objective.

While the nature of DCN traffic is imperfectly understood and certainly varies considerably from one instance to another, there emerge from the literature two characteristics that have a significant impact on the performance of flow scheduling. First, the partition/aggregate structure of datacenter applications implies flows do not occur singly but rather in *batches* and the significant performance indicator is not flow completion time (FCT) but rather batch completion time (BCT), the time to complete every flow in the batch. Second, these batches of flows occur in *bursts*, one batch only beginning after the previous one has completed. This means the arrival process is not independent of the scheduler, a necessary condition for the optimality of SRPT as proved by Schrage [32].

*Work performed as consultant to Huawei

We investigate the impact of batches and bursts using simple queuing systems modelling a single bottleneck link considered in isolation and exactly realizing SRPT or processor sharing (PS) service disciplines. Analysis and simulation is used to illustrate the impact on performance of salient traffic characteristics, like the distribution of batch and burst sizes, on the relative performance of SRPT and PS. A notable theoretical contribution is a derivation of the expected BCT of the M^X/G/1 preemptive shortest job first (PSJF) queue that closely approximates SRPT.

The evaluation confirms that per-batch scheduling, where the link is devoted exclusively to the batch with shortest remaining processing time or is fairly shared between concurrent batches, clearly outperforms per-flow scheduling. The relative performance of batch SRPT and batch PS when batches occur in bursts depends significantly on the assumed traffic characteristics with batch PS outperforming SRPT in some cases.

The considered models are idealizations but their results have practical implications on the design of DCN bandwidth sharing mechanisms. In particular, we suggest fair sharing between batches of flows is a desirable pragmatic objective, being simpler to implement than approximate SRPT and having better BCT performance in some practically relevant cases. A further contribution of this paper is to indicate how such sharing might be realized using a novel "virtual fair scheduling" algorithm that is arguably simple enough to be implemented in high speed DCN switches.

In the next section we discuss DCN traffic characteristics, identifying the batch and burst structure of flow arrivals. In Sec. 3 we consider the M^X/G/1 queue under PSJF, SRPT and PS disciplines while in Sec. 4 we compare SRPT with PS when flow batches arrive in bursts. The use of the novel virtual fair scheduling algorithm to realize per-batch fair sharing is discussed in Sec. 5.

2. DCN TRAFFIC CHARACTERISTICS

There is, of course, no general purpose model of datacenter traffic. In this section we seek only to extract from the literature some salient features that need to be taken into account in evaluating the respective performance of size-based scheduling and fairness.

2.1 Flow arrivals are not Poisson

It is notable that, to our knowledge, there are no published results that show that flows in a DCN (i.e., source-destination flows transferring a single piece of data) ever occur as a Poisson process although this is the assumed traffic model in papers advocating size-based scheduling [5, 27, 28]. Results in frequently cited papers like [10], for instance, actually show the contrary while rare studies like [1] that analyze the arrival process in depth have exhibited self-similarity. In fact, flows do not occur singly but in *batches*, with multiple simultaneous data transfers proceeding in

parallel, and *bursts*, where new batches of flows begin only when the previous batch has completed. The correlation induced by this structure significantly impacts the performance of schedulers.

2.2 Flows occur in batches

Chowdhury and Stoica [11] coined the term *coflow* to describe a collection of flows generated by cluster computing applications. The flows have endpoints in one or more machines and share a common performance goal in that all flows typically need to complete to fulfill that goal. Dogar *et al.* [13] similarly recognized that flows typically occur in batches, notably for web query-like applications where a single request is partitioned among a large set of workers that all respond in a short space of time. Collectively scheduling flows in a coflow has been shown to bring significantly shorter completion times than independent per-flow scheduling [12, 13].

While coflows are prevalent in cluster computing, the impact of flow scheduling on job completion time may not be highly significant since data transfer only counts for a very small fraction of this [30]. Flow scheduling is much more critical for web query-like applications where request processing times are very short and response time depends heavily on network delays. The preponderance of network delays in this context is well-known [2]. Moreover, it is becoming more pronounced as DCNs increasingly adopt remote direct memory access (RDMA) technology, bringing ever smaller processing times, e.g., [25].

Some statistics on query traffic are provided in papers describing the use of memcached in Facebook datacenters [6, 29]. Each user request managed by a web client gives rise to hundreds or thousands of object retrievals from a cluster of cache servers. Each object is around 1 KB but retrievals are grouped together with an average of 24 objects included in a single cache-client flow. The number of flows in a batch here depends on the number of cache servers and is not highly variable.

The size of the batch, also known as the incast degree when the flows use a common link [4], may be much larger than that reported in [29]. For instance, Google reports incast degrees that may be measured in thousands for the BigQuery application [25].

2.3 Flows occur in bursts

Flows in a coflow are by definition independent, in the sense that the input of a flow does not depend on the output of another in the same coflow [12]. Additional correlation in the flow arrival process arises because some cluster computing applications proceed in stages, one stage beginning only when the previous stage is finished. Similarly, a web client will manage user requests sequentially, yielding an arrival process where a new coflow will only occur after the previous one is complete.

The exact burst structure of flow arrivals depends on the specific application and is not well-understood. However, while there is no general and widely accepted burst model, it is commonly agreed that this characteristic is clearly present in DCN traffic. Some cluster computing models discussed in [11], like "bulk synchronous parallel", proceed in supersteps: the coflow of one superstep only begins after a barrier synchronization event. Web queries naturally occur in bursts as each response often leads to a new request while a request may itself generate a sequence of dependent coflows [29]. Web clients may also handle a continuous stream of distinct end-users as requests are dispatched by load balancers to idle clients.

It is worth noting here that the well-known mean completion time optimality of SRPT is, in fact, only proved for arrival sequences that are independent of the service process [32]. If a coflow or batch can only start some time after a previous coflow has ended, it may well be that size-based scheduling is less desirable than other options like fair sharing.

2.4 Queuing models

To gain insight and better understand the impact on performance of the above DCN traffic characteristics, we consider two abstract queuing models. The models relate to an isolated network link (e.g., from the top-of-rack switch (ToR) to a server hosting web clients) receiving flows that arrive in batches and in bursts.

The first applies to a link that receives batches of flows, all starting simultaneously at the instants of a Poisson process: the objective is to compare size-based scheduling and fair sharing under different assumptions regarding the flow size and batch size distributions. The second model accounts for burst arrivals: in the light of results for the first model and previous work on coflow scheduling [12, 13], the model considers successive coflows in the burst as a whole rather than scheduling individual flows.

3. BATCHES OF FLOWS

We seek to compare the expected batch completion time, E[BCT], for the M^X/G/1 system with size-based or fair scheduling. Batches of flows arrive as a Poisson process at rate λ. The number of flows in a batch B is an independent and identically distributed (i.i.d.) random variable. The individual flow size is also i.i.d. with distribution $F(x)$ and density $f(x)$ and, to avoid unhelpful complications, this distribution has no atoms. The link has unit capacity. Following [12], the size of its largest flow is the *batch length* L, the number of flows is the *batch width* B, and the sum of flow sizes is the *batch size* S. In the next three subsections, Sec. 3.1-3.3, flows are scheduled individually without regard to the batch they belong to. The resulting E[BCT] performance is compared numerically in Sec. 3.4 with that realized when batches are scheduled as a whole. We first consider Preemptive Shortest Job First (PSJF) scheduling, that has similar performance to SRPT while being simpler to analyse.

3.1 PSJF

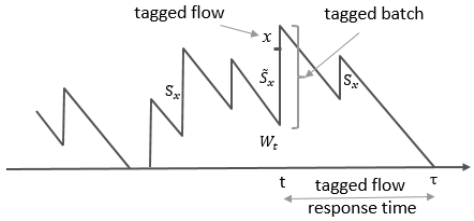
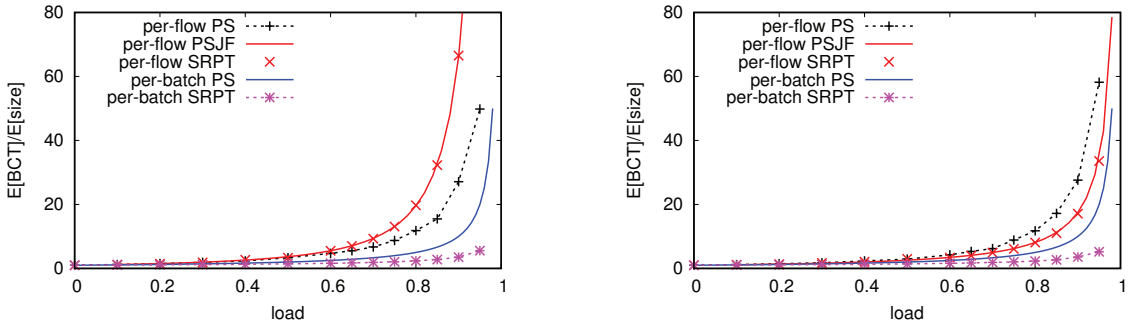


Figure 1: Response time of tagged flow as a busy period.

For M/G/1, the performance of PSJF is close to that of the optimal SRPT [36]. Here we derive E[BCT] for PSJF with batch arrivals. Consider a tagged flow of size x and refer to the batch it arrives in as the tagged batch. The tagged flow response time is equal to a certain residual busy period in a work conserving M/G/1 queue where customers are i.i.d. batches of flows of size $< x$ having combined size S_x, as illustrated in Fig. 1. The residual busy period begins at the tagged batch arrival time, t, when the total work in system is equal to the sum of three components:
i) W_t, the work in system at t due to flows of original size $< x$;
ii) $\tilde{S}_x$, the combined size of flows of size $< x$ in the tagged batch;
iii) the tagged job itself of size x.
It extends until the tagged flow completes at time τ and includes the service time of any batches of flows of size $< x$ arriving between t and τ. Let $m_i(x) = \int_0^x t^i f(t)dt$ be the i^{th} moment of the size of



(a) flow $\text{cv}^2 = 0.1$, batch $\text{cv}^2 = 1$

(b) flow $\text{cv}^2 = 10$, batch $\text{cv}^2 = 1$

Figure 2: Batch arrivals: normalized expected batch completion time ($\text{E}[\text{BCT}]/\text{E}[S]$) as a function of the load. Plots generated with geometric batch width, mean= 100, flow size $\text{CV}^2 = 0.1$ (left) or $\text{CV}^2 = 10$ (right); crosses are from simulations, full lines from analysis.

flows of size $< x$ and let $\rho(x) = \lambda \text{E}[B] m_1(x)$ be the system load due to such flows. The proof of the following theorem is given in the appendix.

THEOREM 1. *For the M^X/G/1 PSJF system, we have,*

$$\text{E}[\text{FCT}] = \int_{x \geq 0} \left(\frac{\text{E}[W_t] + \text{E}[\tilde{S}_x^f] + x}{1 - \rho(x)} \right) f(x) dx,$$

and

$$\text{E}[\text{BCT}] = \int_{x \geq 0} \left(\frac{\text{E}[W_t] + \text{E}[\tilde{S}_x^b] + x}{1 - \rho(x)} \right) g(x) dx,$$

where

$$\begin{aligned} \text{E}[W_t] &= \frac{\lambda(\text{E}[B(B-1)] m_1(x)^2 + \text{E}[B] m_2(x))}{2(1-\rho(x))}, \\ \text{E}[\tilde{S}_x^f] &= (\text{E}[B^2]/\text{E}[B] - 1) m_1(x), \\ \text{E}[\tilde{S}_x^b] &= \frac{\text{E}[B(B-1)F(x)^{B-2}]}{\text{E}[BF(x)^{B-1}]} m_1(x), \end{aligned}$$

and $g(x) = \text{E}[BF(x)^{B-1}] f(x)$ is the density of the tagged batch length.

The following is used later to compute results in Fig. 2.

COROLLARY 1. *If B has a geometric distribution of mean β, the formulas become:* $\text{E}[W_t] = (\beta^2 m_1(x)^2 + \beta m_2(x))/(1 - \rho(x))$, $\text{E}[\tilde{S}_x^f] = 2\beta m_1(x)$, $\text{E}[\tilde{S}_x^b] = 2\beta/(1 + \beta - \beta F(x))$ *and* $g(x) = \beta/(1 + \beta - \beta F(x))^2$.

Numerical experiments with the formulas of Theorem 1 confirm that PSJF performance is highly sensitive to the flow size distribution with E[BCT] decreasing as the flow size variance increases. The decrease is, however, less marked than the corresponding decrease in E[FCT]. When the variance is high, the main difference between E[BCT] and E[FCT] comes from the difference between densities $f(x)$ and $g(x)$ since the conditional batch response times are nearly equal. For lightly varying flow sizes, on the other hand, there is also a significant difference between the own-batch terms $\text{E}[\tilde{S}_x^f]$ and $\text{E}[\tilde{S}_x^b]$ that accentuates the negative impact on E[BCT] of per-flow size-based scheduling (see Sec. 3.4).

3.2 SRPT

To evaluate E[BCT] we can adapt the analysis of M^X/G/1 SRPT of Gebrehiwot *et al.* [15] where an expression for E[FCT] is derived. Their analysis conditions the response time of a tagged flow on four different server states at the arrival instant of the tagged batch. These states remain relevant and the only change in the analysis is to account for the composition of the tagged batch that here has length x while in [15] it just contains an arbitrary tagged job of size x. This is used in calculating "the waiting time of a type-x job caused by jobs in its own batch", denoted $W^b(x)$ in [15].

In the notation of the previous section, the expected combined size of the flows of size $< x$ in the tagged batch is $\text{E}[\tilde{S}_x^b]$ given in Theorem 1 and, $W^b(x) = \text{E}[\tilde{S}_x^b]/(1 - \rho(x))$. This expression can be substituted in the formulas from [15, Th 2] to derive an expression for E[BCT]. This expression is not particularly insightful, however, and numerical evaluation is not straightforward. In our evaluation, we have preferred to use simulation, relying on the analytical results for PSJF to provide insight.

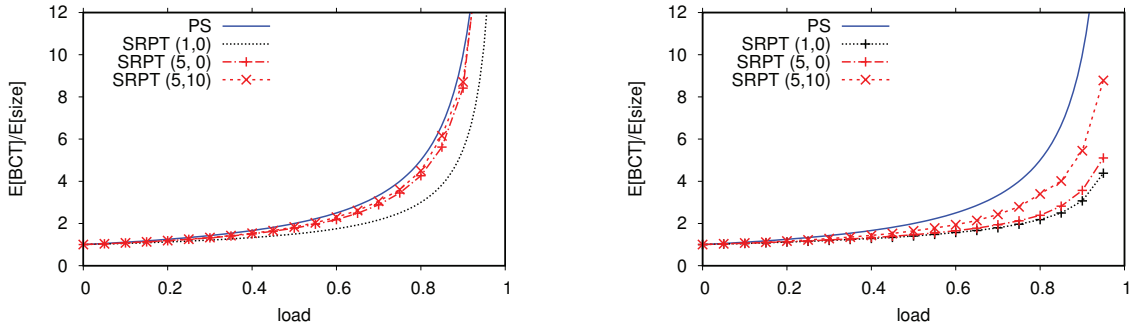
3.3 PS

Unlike the regular M/G/1 PS queue, the batch arrival M^X/G/1 queue with per-flow PS scheduling has no simple and general performance formulas. The integral equation formulation of Kleinrock *et al.* [22] can be solved to derive the conditional response time of an arbitrary flow of size x and consequently E[FCT]. Bansal [8] provides a computational scheme for a class of flow distributions while Avrachenkov *et al.* [7] have derived conditional response time asymptotics. In recent work Guillemin [19] and co-authors derive the Laplace transform of the batch completion time but only for the M^X/M/1 system with geometric batch width. We therefore again rely on simulation to evaluate E[BCT] for the per-flow M^X/G/1 PS system.

3.4 Comparative performance

We compare the E[BCT] performance of M^X/G/1 under per-flow PSJF, SRPT and PS scheduling. We further compare per-flow scheduling with per-batch scheduling for SRPT and PS, i.e., where the server capacity is devoted exclusively to the *batch* with the shortest remaining overall size, and where active *batches* share capacity equally, respectively. For these and later comparative results, we ran the simulations long enough to ensure the presented results are accurate at the scale of the figure.

Fig. 2 plots E[BCT] against load ($\lambda \text{E}[S]$) for two contrasting unit mean flow size distributions, Weibull with shape parameter 3.5 (squared coefficient of variation, $\text{CV}^2 \approx 0.1$) and Weibull with shape parameter 0.4 ($\text{CV}^2 \approx 10$). The number of flows per batch has a geometric distribution of mean 100 (the batch size CV^2 is



(a) batch size cv^2=0

(b) batch size cv^2=10

Figure 3: Partly-open model with homogeneous traffic: normalized expected BCT against load; batch size is deterministic (left) or Weibull with $cv^2 = 10$ (right); mean burst length is 1 (denoted 'SRPT (1,0)') or 5 with $CV^2 = 0$ ('SRPT (5,0)') or $CV^2 = 10$ ('SRPT (5,10)'); exponential inter-batch interval, mean = 1 full rate batch service time.

then only 1.035 and 1.004, respectively). These results illustrate the following qualitative properties (confirmed by results for other parameter values):

- per-flow PSJF has very similar performance to SRPT, the simulation results for SRPT (crosses) being slightly lower than the analytical PSJF results;
- per-flow SRPT is preferable to PS when the size distribution has high variance but has higher E[BCT] when the variance is small[1];
- per-batch scheduling is significantly more effective than per-flow scheduling;
- per-batch PS performs less well than per-batch SRPT (that is optimal [32]).

Per-flow SRPT has poor E[BCT] performance because the largest flow in a batch naturally becomes a straggler, being neglected in favor of shorter flows in concurrent batches. This phenomenon is accentuated as the size distribution becomes more concentrated about the mean. Per-flow PS also creates stragglers since the largest flow in a batch is eventually alone and in competition with multiple, still active flows in concurrent batches. This phenomenon is worse when the size variance is high since the largest flows then typically compete unfairly for a longer period of time. Per-batch scheduling avoids the straggler phenomenon for both SRPT and PS.

4. BURSTS OF FLOWS

In this queuing model, batches of source-destination flows, considered for scheduling as a whole, occur in bursts: when a batch completes, it may end the burst or be followed by a new batch after the lapse of an interval of inactivity. We assume for present purposes that it is possible to perform SRPT or PS at batch level. The issue of realizing batch scheduling in practice is briefly discussed in Sec. 4.5. The considered model is a network of two successive service stations, one representing the link that implements either SRPT or PS scheduling, the other an infinite server representing the inactivity interval. Following Schroeder *et al.* [33] we distinguish a *closed* network model where a fixed number of sources generate successive batches of flows continuously, and a *partly-open* model where bursts arrive as a Poisson process and generate a finite number of successive batches. The case where the "burst" is always of size 1 (i.e., Poisson batch arrivals) is termed an *open* model and was considered in Sec. 3.

4.1 Performance of PS and SRPT

The performance of PS under the partly-open and closed models is well-understood. Average values like E[BCT] are insensitive to the distributions of batch sizes and inactivity intervals that can even be correlated (see [21, Ch. 3], for example). For the partly-open model, $E[BCT] = 1/(1-\rho)$, as for the open model, for any distribution of the number of batches in a burst. Performance of the closed model depends on the number of sources of different types and can be numerically difficult to compute.

SRPT performance depends sensitively on the distributions of batch size and burst length and we are aware of no useful analytical results for partly-open and closed models. To gain understanding of the comparative performance of SRPT and PS scheduling, we have simulated some specific configurations under a traffic model where batch sizes and inactivity intervals are i.i.d. and mutually independent. The following sections illustrate the impact on SRPT performance of some particular choices of parameter values.

4.2 Single class, partly-open model

Figure 3 plots the normalized mean BCT against load for a number of homogeneous partly-open model configurations. The interval between batches is exponential with mean equal to the expected service time of a single batch. We confirmed the observation in [33] that SRPT performance does not depend significantly on the size of the inactivity interval. The figure plots normalized E[BCT] for three burst configurations: bursts of exactly 1 batch (i.e., the open model), bursts of exactly 5 batches, and bursts of 5 batches on average with a hyper-geometric distribution with CV^2 of 10. The legend gives the (mean, CV^2) combination of the bursts in question. The batch size distribution is deterministic on the left and Weibull with CV^2 10 (shape parameter .4) on the right.

PS performance is, of course, the same for all configurations. SRPT has consistently lower E[BCT] than PS but the difference decreases as the burst size and variance increase, especially when the batch size CV^2 is small. Intuitively, as the burst size gets larger, SRPT switches service between concurrent bursts uniformly at ran-

[1]Note that SRPT with a deterministic distribution is the same as FCFS and has better performance than PS; this is a singularity not covered by the present assumption that $F(x)$ has no atoms.

dom so that each tends to get a fair share on average. This explains why we can expect SRPT performance to converge to that of PS, i.e., E[BCT]$\rightarrow 1/(1-\rho)$. SRPT is closer to PS when batch size CV^2 is small.

4.3 Multi-class, partly-open model

When there are multiple classes of bursts with distinct characteristics, SRPT can result in worse overall E[BCT] than PS. This is illustrated by the results of Fig. 4. The figure plots E[BCT] against load when the link receives two classes of traffic: class 1 batches are of deterministic size 1 and arrive in bursts of mean 5 and CV^2 0 on the left, and mean 5 and CV^2 10 on the right; class 2 batches are of deterministic size 2 and arrive individually. Each class counts for half the overall load.

The results show that the unfairness of SRPT leads to relatively poor performance for class 2 batches and that this can yield an overall E[BCT] worse than that of PS when the burst variance is high. To understand the negative impact of the class 1 burst size distribution, consider the case where the inactivity interval is very small. The system then behaves like an M/G/1 preemptive priority queue: low priority class 2 response time increases linearly with the second moment of the combined size of batches in a class 1 burst (e.g., [36, Sec. 3.2.3]).

4.4 Closed model

Fig. 5 presents results for a closed model where a fixed number of 'clients' initiate a succession of batches of flows separated by an exponential inactivity interval. The size of the interval is varied to produce a range of link utilizations.

For the left hand figure, 100 clients emit batches with a Weibull size distribution of CV^2 0.1 or 10. The results confirm the observation in [33] that completion times are smaller in the closed model than in open and partly-open models. SRPT is better than PS and its E[BCT] is nearly insensitive.

The right hand figure illustrates SRPT unfairness when there are two classes of client. The larger class 2 batches are penalized though their normalized E[BCT] is only worse than the PS average when utilization tends to 1. Class 2 clients are eventually starved of any service under SRPT while PS converges to a service rate of 1/100 for all clients.

4.5 Batch scheduling

We have assumed in the above that it is feasible to perform scheduling at batch level for either SRPT or PS. However, it is important to note that while batch PS might be realized in a distributed manner, as discussed in the next section, batch SRPT appears difficult to realize in practice. A central scheduler made aware of the size of all flows in the batch is envisaged in work on coflow scheduling (e.g., [12, 9]) but this is complex and hardly practical for web query-like applications. The approximate SRPT schemes proposed in cited papers [5], [27] and [28] apply to source-destination flows and cannot obviously be adapted to work at batch level. The task scheduler Baraat [13] does have a distributed implementation but essentially realizes an enhanced batch FIFO service scheme rather than SRPT.

We suggest the predictable, insensitive performance of batch PS, that is often as good as batch SRPT or even better (results of this section) and clearly better than per-flow scheduling (results of Sec. 3), make it worth exploring a possible realization (next section). Of course, there is scope for improved performance in certain cases (e.g., where the closed model of Sec. 4.4 would apply) and continued research on the design of practical, size-based schedulers operating at batch level remains valuable.

5. REALIZING PER-BATCH FAIR SHARES

The evaluation in Sec. 3 shows it is advantageous to perform per-batch rather than per-flow scheduling. The purpose of this section is to suggest how per-batch *fairness* might be realized in the datacenter by means of a novel "virtual fair scheduling" algorithm. In this section we use 'flow' in the general sense meaning all packets having in common a certain set of header fields. For *batch* fairness, these fields should identify the specific client in the server receiving the batch. However, the proposed VFS algorithm could also be applied to enforce fairness between flows defined by some other criterion.

5.1 End-system and switch mechanisms

To realize per-batch fairness it is necessary to implement mechanisms in both switch and end-systems. Fairness on the ToR-server link would be enforced by a switch mechanism supposed able to recognize packets belonging to a given batch from header fields identifying the client having initiated multiple parallel requests. The end-systems are supposed to implement a transport protocol to manage the individual flows of the batch and efficiently exploit the enforced fair bandwidth share.

The switch might implement a classical fair queuing scheduler like deficit round robin [35] or start-time fair queuing [18]. The number of batches that need to be scheduled in the open or partly-open models discussed in Sec. 4 has a geometric distribution when fairness is enforced and per-batch scheduling is therefore scalable and feasible [23], e.g., the number of active flows at load .9 is less than 66 with probability .999. The scalability of fair queuing has been confirmed in recent work where approximate realizations of the classical scheduling algorithms have been proposed [34], [16], [17]. To implement approximate fair queuing in high speed datacenter switches remains challenging, however. Our objective here is to propose a much simpler algorithm, that we call virtual fair scheduling (VFS), as an attractive alternative.

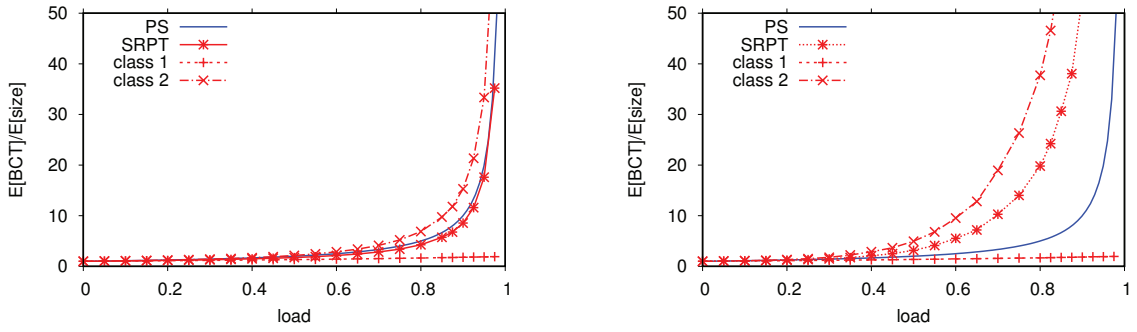
In the next section we present the simplest version of this algorithm that enforces fairness by dropping excess packets. Fair dropping is well-known to be a viable means to enforce per-flow fairness [31], [3]. End systems are supposed to interpret drops as congestion signals and react appropriately by reducing flow rates, as in classical TCP/IP. We do not seek here to further define the required transport protocol. We just assume it efficiently distributes the batch fair bandwidth share among flows in a work-conserving manner (e.g., by adapting protocols in [14] or [27]) .

5.2 Virtual fair scheduling

VFS is a significant simplification of the fair dropping algorithm proposed by Addanki *et al.* [3]. While that algorithm arguably enforces *exact* fair shares, it is much too complex to be implemented in a datacenter switch. VFS on the other hand, is less precise but much simpler. VFS pseudocode is shown in Algo. 1. Note that VFS updates state for only one flow at a time and employs a simple round robin-based management of the active flow list. Its simple operation and lean data structure arguably make VFS readily implementable in a modern, high speed DCN switch.

As in the fair dropping algorithm of [3], VFS implements a virtual scheduler that determines when flows exceed the fair rate and packets need to be dropped. Packets are actually scheduled in a simple FIFO that is assumed large enough that the probability of saturation is negligibly small given that flow rates are controlled by VFS.

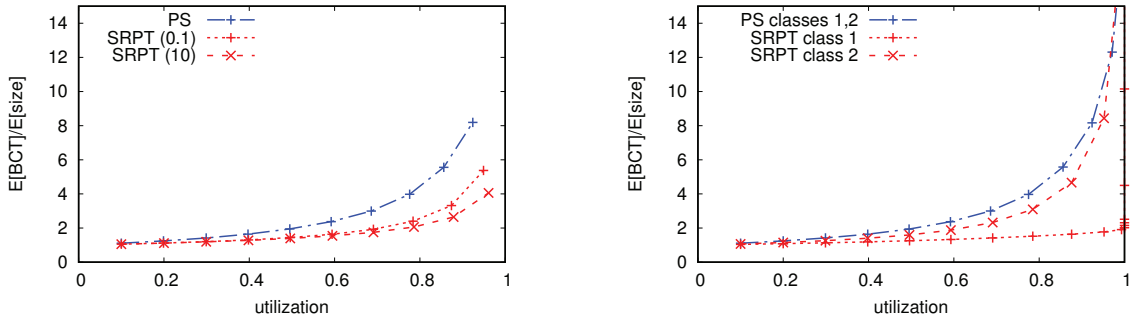
The algorithm relies on a table of active flows $\mathcal{A}$ that records the current occupancy in bytes, $flow.vq$, of a dedicated virtual queue (VQ) for each flow in $\mathcal{A}$. This occupancy is incremented on packet



(a) burst mean 5, cv^2 0

(b) burst mean 5, cv^2 10

Figure 4: Partly-open model with two classes of traffic: normalized expected BCT against load; class 1 batches of deterministic size 1 arrive in bursts of mean length 5 and $CV^2 = 0$ (left) or $CV^2 = 10$ (right), exponential inter-batch interval, mean = full rate batch service time; class 2 batches of deterministic size 2 arrive singly; each class contributes half the load.



(a) 100 homogeneous clients

(b) 50 class 1 clients, 50 class 2 clients

Figure 5: Closed model with one (left) and two classes of traffic (right): normalized expected BCT against utilization; left – 100 clients generate batches with size 1 and CV^2 0.1 or 10; right – 50 class 1 clients generate batches of size 1 and CV^2 0.1, 50 class 2 clients generate batches of size 2 and CV^2 .1; the interval between batches is modulated to generate the range of utilizations keeping the ratio E[batch size]/E[interval] the same for each class.

arrival, unless the counter already exceeds a threshold θ in which case the packet is dropped, and decremented at certain epochs, as explained in the next paragraph. Given the scalability of fair queuing, the flow table $\mathcal{A}$ can be realized efficiently as a hash table, as proposed in [34] or [16] for example.

The elements of $\mathcal{A}$ are arranged in a linked list and, at a series of epochs s_m for $m = 1, 2, \ldots$, the VQ of the flow currently at the head of this list is decremented. The decrement is equal to the minimum of a state variable $credit$ and the present value of $flow.vq$. Variable $credit$ gives the amount of service capacity that is so far unallocated to any flow. It is increased at every arrival (line 2). If the chosen VQ is reduced to zero, that flow is removed from $\mathcal{A}$ while the residual credit is retained and used at the next epoch s_{m+1} to decrement the next VQ in the linked list. The flow at the head of the linked list steps in round robin order on each epoch s_m.

This algorithm is fair in the sense that any two flows that are permanently backlogged in the shadow system in some interval receive the same *expected* amount of service capacity. The variance of the service amount each receives depends on the relative frequency of the epochs s_m. We experimented with this and found a frequency equivalent to the arrival rate (e.g., decrementing the VQ of the head of $\mathcal{A}$ immediately after incrementing the VQ of the flow of the arriving packet) is sufficient. However, it is not necessary that $s_n = t_n$ for all n and it can be advantageous for practical reasons to set the s_m differently.

5.3 VFS performance

Assuming the algorithm does enforce fair shares like a PS system, we would expect the normalized E[FCT] at load ρ to be $1/(1-\rho)$. Moreover, the distribution of the number of flows in the active list would be geometric with a p-percentile equal to $\log(1 - p/100)/\log(\rho)$. Figure 6 confirms that this is true for a particular traffic model with VQs decremented following each arrival, i.e., $s_m = t_m$.

The figure plots simulation results as crosses and the analytical PS results as lines, for E[FCT] (left y-axis) and the 99.9 percentile of the distribution of $|\mathcal{A}|$ (right y-axis), against load ρ. The traffic is composed of a Poisson process of flows emitting at line rate until 3000 packets have been successfully transmitted together with a Poisson process of single packet flows. The traffic volume in bit/s is 80% due to the large flows and 20% due to singletons. The results confirm that the analysis accurately predicts the simulation results for this particular case. We have verified for numerous other configurations that the agreement is true in general.

Algorithm 1 Virtual fair scheduling

1: {On arrival of n^{th} packet at time t_n}
2: $credit \mathrel{+}= C(t_n - t_{n-1})$
3: input pkt of $flow(pkt)$
4: **if** $flow \in \mathcal{A}$ **then**
5: **if** $flow.vq > \theta$ **then**
6: drop pkt
7: **else**
8: $flow.vq \mathrel{+}= pkt.length$
9: **end if**
10: **else**
11: $\mathcal{A} = \mathcal{A} \cup flow$
12: $flow.vq = packet.length$
13: add $flow$ as tail($\mathcal{A}$)
14: **end if**
 {At some epochs s_m, eg, $s_n = t_n$}
15: $flow = head(\mathcal{A})$
16: **if** $credit < flow.vq$ **then**
17: $flow.vq \mathrel{-}= credit$
18: $credit = 0$
19: **else**
20: $credit \mathrel{-}= flow.vq$
21: $\mathcal{A} = \mathcal{A} \setminus flow$
22: **end if**
23: **if** $|\mathcal{A}| > 0$ **then**
24: $head(\mathcal{A}) = next$
25: **end if**

5.4 VFS enhancements

The algorithm can be enhanced in several ways. It is possible to ensure low packet latency for flows emitting at a rate less than the fair rate by forwarding their packets via a priority queue [24, 20]. These packets are identified by the fact that their flow is not already present in $\mathcal{A}$ when they arrive.

The algorithm can be adapted to provide weighted fair shares. Instead of following the linked list round robin order (line 15), one could select the VQ to be decremented with a probability proportional to its weight.

A more significant enhancement would be to replace packet drops as a congestion signal by an explicit notification. We have in mind feeding back to the end systems a measure of the currently realized fair rate. This would allow the end systems to pace packet emissions to avoid loss and delay, thus realizing the goal of lossless networking, as discussed in [26] and [25], for example. The fair rate can be derived by measuring the load due to packets from non-backlogged flows ($flow \notin \mathcal{A}$ on packet arrival), subtracting this from the link capacity C, and dividing by $|\mathcal{A}|$.

We have verified the feasibility of these enhancements with small-scale simulations of a single link. It remains, however, to perform a more thorough evaluation and to experimentally validate a possible implementation in a high speed switch.

6. CONCLUSION

By evaluating two abstract queuing systems, we have thrown light on the impact on performance of two particularities of DCN traffic: (i) flows frequently occur in batches due to the use of the partition/aggregate paradigm, and (ii) flow batches occur in bursts where a new batch only starts after the previous one has completed. This structure significantly impacts the performance of scheduling algorithms like SRPT and PS and challenges the supposed superiority of the former, previously demonstrated in the literature under an unrealistic Poisson flow arrivals traffic model [5, 27, 28].

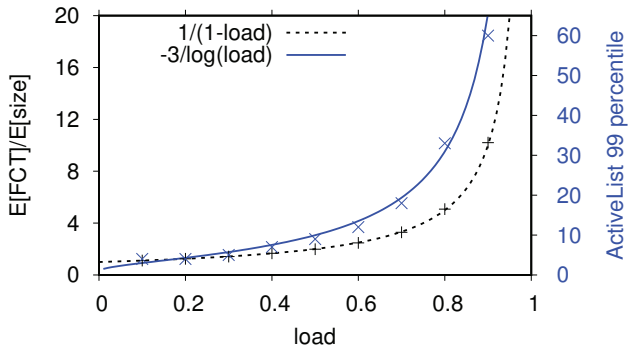


Figure 6: Performance of VFS: simulations (crosses) compared to analysis (lines); normalized E[FCT], $1/(1-\rho)$, and 99.9 percentile of active list size, $-3/\log_{10}\rho$.

Analytical and simulation results in Sec. 3 confirm that per-batch scheduling, where the link is devoted exclusively to the batch with shortest remaining processing time (batch SRPT) or is fairly shared between concurrent batches (batch PS), clearly outperforms per-flow scheduling. While per-batch SRPT minimizes BCT for the M^X/G/1 queue, per-flow SRPT can be particularly unfavorable and worse than per-flow PS when the flow size distribution has a small variance.

The superiority of (per-batch) SRPT over PS is significantly mitigated, and can even disappear, when accounting for the burst structure of flow arrivals, as shown in Sec. 4. In particular, SRPT is hardly better than PS when the distribution of flow size has low variance and the number of flows in a burst is large. Additionally, SRPT is unfair to larger flows and this can lead to higher overall expected BCT than PS in certain conditions. While SRPT would be preferable for the closed model, where a finite number of clients receive a never-ending sequence of flow batches, PS performance is not bad and has the advantage of being predictable given the insensitivity of the underlying queuing model.

In light of the above results, we believe per-batch fairness is a reasonable scheduling objective, especially as this appears much simpler to implement than per-batch SRPT. We have suggested how this objective might be realized using a novel "virtual fair scheduling" algorithm, that closely approximates fair queuing while being simple enough to be implemented in a high speed DCN switch.

The presented numerical results clearly cover only a tiny fraction of the parameter space of the considered queuing systems. Moreover, these systems are greatly simplified models of a DCN link and the applied traffic models are idealized representations of the actual flow arrival process. However, we believe the results do usefully highlight the negative impact of the batch and burst structure of DCN traffic on the performance of size-based scheduling algorithms like per-flow SRPT. Designers of schedulers and transport protocols need to be aware of this impact and carefully evaluate network performance under a realistic model of its traffic.

Appendix: M^X/G/1 PSJF

Proof of Theorem 1 Applying the standard formula for the expected residual busy period of an M/G/1 of load $\rho(x)$ and initial

work $W_t + \tilde{S}_x + x$ (e.g., [36]), we deduce the mean conditional response time,

$$T(x) = \frac{\mathrm{E}[W_t] + \mathrm{E}[\tilde{S}_x] + x}{1 - \rho(x)}.$$

By PASTA, $\mathrm{E}[W_t]$ is the expected work in an M/G/1 queue with load $\rho(x)$ and customer service time S_x and is given by the Pollaczek-Khinchin formula,

$$\mathrm{E}[W_t] = \frac{\lambda M_2(x)}{2(1-\rho(x))},$$

where $M_2(x)$ is the second moment of S_x. Introduce the batch size distribution $\pi_b = \Pr[B = b]$ and denote by X_i, for $1 \le i \le b$, the sizes of flows in a batch of size b. Then,

$$\begin{aligned} M_2(x) &= \mathrm{E}[S_x^2] = \sum_b \mathrm{E}[(\sum_{1 \le i \le b} X_i 1_{X_i < x})^2]\pi_b \\ &= \mathrm{E}[B(B-1)]m_1(x)^2 + \mathrm{E}[B]m_2(x), \end{aligned}$$

where $m_i(x) = \int_0^x t^i f(t)dt$.

The mean tagged batch size $\mathrm{E}[\tilde{S}_x]$ depends on the nature of the tagged flow. To compute E[FCT], the tagged flow is an arbitrary flow of size x and $\tilde{S}_x$, here denoted $\tilde{S}_x^f$, is the combined size of flows of size $< x$ in the tagged batch. The tagged batch size has the distribution, $\pi'_b = b\pi_b/\mathrm{E}[B]$ (since the probability the tagged flow is included in a batch of size b is proportional to b) and, reasoning as above, we have,

$$\begin{aligned} \mathrm{E}[\tilde{S}_x^f] &= \sum_b \mathrm{E}[(\sum_{i \le b-1} X_i 1_{X_i < x})]\pi'_b \\ &= (\mathrm{E}[B^2]/\mathrm{E}[B] - 1)m_1(x). \end{aligned}$$

For the batch completion time, the tagged flow is the largest flow in its batch and $\tilde{S}_x$, denoted $\tilde{S}_x^b$, is the combined size of all other flows in the batch. Recall that the largest flow in a batch is the batch length L and the combined batch size is denoted S. L has distribution $\sum_b F(x)^b \pi_b$ and, therefore, density,

$$g(x) = \sum_b b f(x) F(x)^{b-1} \pi_b = \mathrm{E}[BF(x)^{B-1}]f(x).$$

The expected batch size given $L = x$ and $B = b$ is,

$$\begin{aligned} \mathrm{E}[S | L = x \text{ and } B = b] &= x + \sum_{i \le b-1} \mathrm{E}[X_i | X_i < x] \\ &= x + (b-1)m_1(x)/F(x), \end{aligned}$$

while the batch width B, conditioned on its length being x has the distribution,

$$\Pr[B = b | L = x] = \frac{bF(x)^{b-1}\pi_b}{\mathrm{E}[BF(x)^{B-1}]}.$$

We can now derive the expected combined size of flows of size $< x$ in the tagged batch as,

$$\begin{aligned} \mathrm{E}[\tilde{S}_x^b] &= \sum_b \mathrm{E}[S | L = x \text{ and } B = b] \Pr[B = b | L = x] - x \\ &= \frac{\mathrm{E}[B(B-1)F(x)^{B-2}]}{\mathrm{E}[BF(x)^{B-1}]} m_1(x). \end{aligned}$$

7. REFERENCES

[1] C. L. Abad, N. Roberts, Y. Lu, and R. H. Campbell. A storage-centric analysis of mapreduce workloads: File popularity, temporal locality and arrival patterns. In *IEEE International Symposium on Workload Characterization (IISWC)*, pages 100–109, 2012.

[2] D. Abts and B. Felderman. A guided tour of data-center networking. *Communications of the ACM*, 55(6):44–51, 2012.

[3] V. Addanki, L. Linguaglossa, J. Roberts, and D. Rossi. Controlling software router resource sharing by fair packet dropping. In *Proceedings of IFIP Networking*, 2018.

[4] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan. Data center TCP (DCTCP). In *Proceedings of SIGCOMM 2010*, pages 63–74, 2010.

[5] M. Alizadeh, S. Yang, M. Sharif, S. Katti, N. McKeown, B. Prabhakar, and S. Shenker. pFabric: Minimal near-optimal datacenter transport. In *Proceedings of the ACM SIGCOMM 2013*, pages 435–446, 2013.

[6] B. Atikoglu, Y. Xu, E. Frachtenberg, S. Jiang, and M. Paleczny. Workload analysis of a large-scale key-value store. In *Proceedings of SIGMETRICS/PERFORMANCE*, pages 53–64, 2012.

[7] K. Avrachenkov, U. Ayesta, and P. Brown. Batch arrival processor-sharing with application to multi-level processor-sharing scheduling. *Queueing Syst. Theory Appl.*, 50(4):459–480, Aug. 2005.

[8] N. Bansal. Analysis of the M/G/1 processor-sharing queue with bulk arrivals. *Operations Research Letters*, 31(5):401–405, 2003.

[9] C. Benet, A. Kassler, G. Antichi, T. Benson, and G. Pongracz. Providing in-network support to coflow scheduling. In *IEEE NetSoft*, 2021.

[10] T. Benson, A. Akella, and D. A. Maltz. Network traffic characteristics of data centers in the wild. In *Proceedings of IMC âĂŹ10*, pages 267–280, 2010.

[11] M. Chowdhury and I. Stoica. Coflow: A networking abstraction for cluster applications. In *Proceedings of HotNets-XI*, pages 31–36, 2012.

[12] M. Chowdhury, Y. Zhong, and I. Stoica. Efficient coflow scheduling with Varys. In *Proceedings of SIGCOMM '14*, pages 443–454, 2014.

[13] F. R. Dogar, T. Karagiannis, H. Ballani, and A. Rowstron. Decentralized task-aware scheduling for data center networks. In *Proceedings of SIGCOMM '14*, pages 431–442, 2014.

[14] 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 CoNEXT '15*, 2015.

[15] M. E. Gebrehiwot, S. Aalto, and P. Lassila. Energy-aware SRPT server with batch arrivals. *Perform. Eval.*, 115(C):92–107, Oct. 2017.

[16] P. Goyal, P. Shah, N. K. Sharma, M. Alizadeh, and T. E. Anderson. Backpressure flow control. In *Proceedings of the 2019 Workshop on Buffer Sizing*, 2019.

[17] P. Goyal, P. Shah, K. Zhao, G. Nikolaidis, M. Alizadeh, and T. E. Anderson. Backpressure flow control. In *Proceedings of NSDI 22*, pages 779–805, 2022.

[18] P. Goyal, H. Vin, and H. Cheng. Start-time fair queueing: a scheduling algorithm for integrated services packet switching networks. *IEEE/ACM Transactions on Networking*, 5(5):690–704, 1997.

[19] F. Guillemin, A. Simonian, R. Nasri, and V. Q. Rodriguez.

Asymptotic analysis of the sojourn time of a batch in an $M^{[X]}/M/1$ processor sharing queue. arXiv, 2021.

[20] T. Hoeiland-Joergensen, P. McKenney, D. Taht, J. Gettys, and E. Dumazet. The Flow Queue CoDel Packet Scheduler and Active Queue Management Algorithm. RFC 8290, Jan. 2018.

[21] F. P. Kelly. *Reversibility and Stochastic Networks*. Wiley, 1979.

[22] L. Kleinrock, R. Muntz, and E. Rodemich. The processor-sharing queueing model for time-shared systems with bulk arrivals. *Networks*, 1:1–13, 1971.

[23] A. Kortebi, L. Muscariello, S. Oueslati, and J. Roberts. Evaluating the number of active flows in a scheduler realizing fair statistical bandwidth sharing. In *ACM SIGMETRICS*, 2005.

[24] A. Kortebi, S. Oueslati, and J. Roberts. Implicit service differentiation using deficit round robin. In *Proceedings of ITC19*, 2005.

[25] G. Kumar, N. Dukkipati, K. Jang, H. M. G. Wassel, X. Wu, B. Montazeri, Y. Wang, K. Springborn, C. Alfeld, M. Ryan, D. Wetherall, and A. Vahdat. Swift: Delay is simple and effective for congestion control in the datacenter. In *Proceedings of SIGCOMM '20*, pages 514–528, 2020.

[26] Y. Li, R. Miao, H. H. Liu, Y. Zhuang, F. Feng, L. Tang, Z. Cao, M. Zhang, F. Kelly, M. Alizadeh, and M. Yu. HPCC: High precision congestion control. In *Proceedings of SIGCOMM '19*, pages 44–58, 2019.

[27] B. Montazeri, Y. Li, M. Alizadeh, and J. Ousterhout. Homa: A receiver-driven low-latency transport protocol using network priorities. In *Proceedings of Sigcomm*, pages 221–235, 2018.

[28] A. Mushtaq, R. Mittal, J. McCauley, M. Alizadeh, S. Ratnasamy, and S. Shenker. Datacenter congestion control: Identifying what is essential and making it practical. *SIGCOMM Comput. Commun. Rev.*, 49(3):32–38, Nov. 2019.

[29] R. Nishtala, H. Fugal, S. Grimm, M. Kwiatkowski, H. Lee, H. C. Li, R. McElroy, M. Paleczny, D. Peek, P. Saab, D. Stafford, T. Tung, and V. Venkataramani. Scaling memcache at Facebook. In *Proceedings of NSDI '13*, pages 385–398, 2013.

[30] K. Ousterhout, R. Rasti, S. Ratnasamy, S. Shenker, and B.-G. Chun. Making sense of performance in data analytics frameworks. In *Proceedings of NSDI'15*, pages 293–307, 2015.

[31] R. Pan, L. Breslau, B. Prabhakar, and S. Shenker. Approximate fairness through differential dropping. *SIGCOMM Comput. Commun. Rev.*, 33(2):23–39, Apr. 2003.

[32] L. Schrage. A proof of the optimality of the shortest remaining processing time discipline. *Operations Research*, 16(3):687–690, 1968.

[33] B. Schroeder, A. Wierman, and M. Harchol-Balter. Open versus closed: A cautionary tale. In *NSDI*, page 18, 2006.

[34] N. K. Sharma, A. Kaufmann, T. Anderson, A. Krishnamurthy, J. Nelson, and S. Peter. Evaluating the power of flexible packet processing for network resource allocation. In *NSDI 17*, pages 67–82, 2017.

[35] M. Shreedhar and G. Varghese. Efficient fair queuing using deficit round-robin. *IEEE/ACM Transactions on Networking*, 4(3):375–385, 1996.

[36] A. Wierman. *Scheduling for today's computer systems: bridging theory and practice*. PhD thesis, Carnegie Mellon University, 2007.