SRPT-based Congestion Control for Flows with Unknown Sizes

Alex Davydow[1], Sergey Nikolenko[1], Vitalii Demianiuk[2], Pavel Chuprikov[3], and Kirill Kogan[2,†]
[1]Steklov Institute of Mathematics at St. Petersburg, Russia [2]Ariel University, Israel [3]USI Lugano, Switzerland

***Abstract*—Modern datacenter transports are required to support latency constraints, usually represented by various forms of flow completion time (FCT). Most implemented congestion control mechanisms that minimize FCT are based on SRPT priorities (e.g., *pFabric* and *Homa*). However, SRPT-based scheduling requires prior knowledge of flow sizes, making this discipline problematic in general. Non-SRPT-based alternatives such as LAS and PIAS are able to cope with this level of uncertainty but suffer from their own limitations: LAS can lead to significant starvation of concurrent elephant flows, while PIAS requires a centralized entity for correct settings. In this work, we generalize SRPT-based scheduling to allow flows with known and unknown sizes to sojourn at the same time. We not only show analytic properties of this generalization but rigorously prove important properties of non-SRPT alternatives with competitive analysis. Based on the proposed SRPT generalization, we introduce a new *ASCC* congestion control. Our main goal is not to propose yet another congestion control but to identify preferable and pathological traffic patterns with unknown flow sizes for various scheduling disciplines. Our observations are validated by an extensive evaluation study.**

I. Introduction

Efficient processing of interactive workloads in datacenters imposes heavy constraints on response time in datacenter fabrics. As a result, traditional datacenter fabrics based on TCP cannot satisfy latency constraints for short flows since they can be affected by coexisting large flows which are not necessarily as sensitive to delays [1, 2]. Multiple datacenter transports have been proposed with the explicit purpose of addressing this constraint [1–8]. Most of these transports optimize various forms of *flow completion time* (FCT) [9]; for instance, *average* FCT is used in [6, 10, 11], *average slowdown* (average normalized FCT) in [5, 8, 11], *tail* FCT in [12, 13].

While there is a consensus in the research community about FCT as the optimized objective, the characteristics that should be incorporated into final decisions depend on specific applications, availability of flow information, and allowed flexibility in buffer management inside switches. Many recently proposed congestion control mechanisms exploit *shortest remaining processing time* (SRPT) [5, 7, 8, 14]. According to [11], it is *easy* to achieve "near-optimal" performance with SRPT-based scheduling disciplines as long as the rate-setting algorithm at the endpoints is reasonable. Unfortunately, SRPT requires prior knowledge of per-flow sizes, which can be problematic in practice, e.g., in the case of HTTP chunked transfers [15] (where dynamically created content is transferred during the generation process), stream processing systems such as *Apache Storm* [16], query response in database systems such as *Microsoft SQL Server* [17], and many others. In general, flow size may become available only with the last packet.

There are several approaches dealing with this level of uncertainty. *Least attained service* (LAS) [18] is an alternative to SRPT-based scheduling that processes flows according to *transmitted* rather than remaining flow sizes. Unfortunately, LAS-based scheduling can lead to significant starvation of large flows that are concurrently active on the same path. For instance, consider two elephant flows arriving to the same switch and assume that the second flow arrives only after the transmission of 99% packets in the first flow. In this case, LAS sends 99% packets in the second flow before it finishes transmission of the first flow. Note that this problem does not exist in SRPT-based scheduling.

Another alternative to SRPT approximately exploiting transmitted flow sizes is PIAS [6]. PIAS maintains a set of thresholds $\alpha_1, \alpha_2, \ldots \alpha_k$ and schedules flows according to the largest index of a threshold that does not exceed the transmitted flow size. Such scheduling allows to use a fixed number of priorities inside switches and can reduce starvation of concurrent elephant flows. Note that PIAS requires a centralized entity collecting information about the flow size distribution to set the thresholds properly. Although non-SRPT-based scheduling disciplines such as LAS and PIAS are valid options allowing to deal with unknown flow sizes, in this work we generalize SRPT to allow flows with known and unknown sizes to sojourn simultaneously. The key idea of proposed SRPT generalizations is to prefer transmitting flows with known sizes first, and transmit these flows according to SRPT priority.

Our contribution is two-fold: (1) we provide analytic results for generalized SRPT-based scheduling and other alternatives that do not exploit flow sizes; (2) we present a comprehensive evaluation study identifying preferable and pathological traffic patterns for congestion control mechanisms based on the analyzed scheduling disciplines. Our analytic results use *competitive analysis* [19] that provides theoretical guarantees in terms of the worst-case factor by which a given algorithm is worse than the optimal offline (clairvoyant) algorithm. In particular, we show that generalized SRPT is 2-competitive and LAS is 3-competitive, while the competitiveness of PIAS

† Dr. Kirill Kogan died on March 13, 2021 from complications of the COVID-19 disease.

$ The work of Alex Davydow and Sergey Nikolenko shown in Sections III, IV, and V was supported by the Russian Science Foundation grant no. 17-11-01276. The work of Vitalii Demianiuk and Kirill Kogan was supported in part by the Israeli Innovation Authority under the Knowledge Transfer Commercialization Program (MAGNETON) file no. 71249, in part by the Ariel Cyber Innovation Center in cooperation with the Israel National Cyber Directorate in the Prime Minister's Office. The work of Pavel Chuprikov was supported by Swiss National Science Foundation, grant #200021 192121, FORWARD.

ISBN 978-3-903176-39-3© 2021 IFIP

and FIFO is not bounded by any constant; to the best of our knowledge, this is the first work providing competitive analysis of these scheduling policies in the general case. In addition, we develop a universal transformation that, for a given policy with proven performance guarantees (upper competitiveness bound) that relies on prior knowledge of flow sizes, constructs a policy that operates on flows with unknown sizes and carries over the competitiveness bound with only a constant factor 3. This theoretical analysis allows to identify the limitations of proposed policies. Next, based on SRPT generalizations, we introduce *agnostic to flow size SRPT-based congestion control* (*ASCC*), a new congestion control mechanism that we compare with PIAS, LAS and SRPT-based congestion controls. Unlike PIAS, *ASCC* does not require a central entity collecting traffic statistics. The goal of this work, however, is not to simply propose a new congestion control mechanism that works better on specific workloads but to develop a general understanding what is achievable in the harsh but realistic environment where flow sizes are uncertain.

II. Big switch abstraction

Similar to near-optimal network designs [5, 8, 14], we represent the datacenter fabric as one big switch S interconnecting servers. In this abstraction, ingress ports correspond to NICs and egress ports to last-hop TOR switches; each ingress port has multiple flows going to various egress ports. We consider various scheduling policies on S that do not require prior knowledge of flow sizes, and we prove their worst-case latency performance guarantees. Note that we use the virtual switch abstraction only for theoretical analysis and algorithmic descriptions. Based on our analytic results, in Section VII we propose congestion control *ASCC* approximating the performance of proposed scheduling policies on S. Note that the transition from an abstract switch model to real congestion control is not immediate and should be extended due to rate control dynamics, imperfect load-balancing, etc.

Let $\mathcal{F} = \{f_1, f_2, \ldots, f_{|\mathcal{F}|}\}$ be a set of flows that are transmitted by S, and let in_i and out_i denote respectively the ingress and egress ports of a flow $f_i \in \mathcal{F}$. A flow f consists of packets $p_1, p_2, \ldots p_l$, where l is the *size* of f and j is the *packet number* of p_j. To simplify exposition, we assume that all packets have uniform size. We assume that time is slotted. Each time slot consists of the *arrival* phase and *transmission* phase. During the arrival phase, packets of a flow $f_i \in \mathcal{F}$ may arrive to S; these packets are assigned to the ingress port in_i. Packets in each flow are numbered in the order of their arrival to S. During the transmission phase, a scheduling policy (algorithm) A in S selects flows in $\mathcal{F}$ that have packets on their ingress ports under the constraint that selected flows cannot share either input or output ports; then A transfers the first untransmitted packet of each selected flow f_i from its ingress port in_i to its egress port out_i. Let S^A be an instance of S managed by a scheduling policy A. Scheduling policies differ in how flows are chosen for transmission. In the simplest case, the big switch abstraction proposed below contains only one ingress port; we denote such a switch by S_1.

The *arrival time* of a flow packet is the time when it arrived to S; the *arrival time* a_i of a flow $f_i \in \mathcal{F}$ is the arrival time of its first packet. We denote by e_i^A the *end time* when A transmits the last packet of f_i from in_i to out_i. A flow f_i is *active* in S^A at time t if $a_i \le t \le e_i^A$. Arrival times of flow packets are unknown to the scheduling policy. We say that A is an *oblivious online* policy if it can use the size of a flow only after its last packet has arrived to S. In this work, we design efficient oblivious online scheduling policies.

The *remaining processing time* $rpt(f)$ of a flow f is the difference between the size of f and the packet number of the first packet in f that still resides on the ingress port of f. SRPT is a policy for S_1 transmitting a packet from a flow f with the smallest value of $rpt(f)$ on every time slot. Note that SRPT is not an oblivious online policy since it requires prior knowledge of the size of f immediately after the arrival of its first packet to S_1. In Section V we propose oblivious online modifications of SRPT-based policies.

In this work, we show worst-case latency performance guarantees of scheduling policies by means of *competitive analysis*. Namely, let $X(A, \mathcal{F})$ be a metric representing the total latency of all flows $\mathcal{F}$ in the switch S^A; in Section III we discuss examples of such metrics. We define OPT_X as an optimal offline algorithm minimizing X for any $\mathcal{F}$; OPT_X is clairvoyant, i.e., it knows flow sizes and arrival times of flow packets in advance. A scheduling policy A is *k-competitive* w.r.t. X if $X(A, \mathcal{F}) \le k \cdot X(\mathrm{OPT}_X, \mathcal{F})$ for any $\mathcal{F}$.

III. Latency representations

The current de-facto standard latency representation in data center transport is the average flow completion time [5, 8]. For a given set of flows $\mathcal{F}$ and a scheduling policy A, the *average flow completion time* (AFCT) is the average time during which flows from $\mathcal{F}$ are active in the switch S^A: $\mathrm{AFCT}(A, \mathcal{F}) = \frac{1}{\mathcal{F}} \sum_{i=1}^{|\mathcal{F}|} \left(e_i^A - a_i + 1\right)$. Below, for ease of exposition we use the *total flow completion time* $\mathrm{FCT}(A, \mathcal{F}) = |\mathcal{F}| \cdot \mathrm{AFCT}(A, \mathcal{F})$; it differs from $\mathrm{AFCT}(A, \mathcal{F})$ by a multiplicative factor that does not depend on the algorithm A and hence does not affect competitive analysis.

We say that a flow f_i is a *gapless* flow if the arrival time of the jth packet p_j from f_i is less than $a_i + j$ for any j (i.e., S receives packets of f_i without gaps). It is known that for a switch S_1 containing a single ingress port, the SRPT policy is optimal w.r.t. FCT if flows are gapless. In the general case, SRPT is not necessary optimal. Moreover, Theorem 1 shows that any scheduling policy for S_1 (including non-oblivious online SRPT) does not have a constant competitive factor w.r.t. FCT in the general case (see Appendix for the proof).

Theorem 1. *For any $n > 0$ and any scheduling policy A managing S_1, there exists a set of flows $\mathcal{F}$ such that* $\mathrm{FCT}(A, \mathcal{F}) \ge \frac{n+1}{3} \cdot \mathrm{FCT}(\mathrm{OPT}_{\mathrm{FCT}}, \mathcal{F})$.

The main reason for this negative result is that A does not know arrival times of packets in advance. To obtain latency performance guarantees of a scheduling policy A, we represent the total latency of flows in $\mathcal{F}$ by the total *flow processing*

time FPT defined as follows: $\mathrm{FPT}(A,\mathcal{F}) = \sum_{i=1}^{|\mathcal{F}|} e_i^A$. The FPT metric is less sensitive than FCT to the differences in arrival times of packets from the same flow, and hence, allows to obtain worst-case latency performance guarantees for considered policies. In this work, we will propose scheduling policies with a constant competitive factor w.r.t. FPT.

Theorem 2. *For any set of flows $\mathcal{F}$, a scheduling policy A outperforms A' w.r.t.* FPT *if and only if A outperforms A' w.r.t.* FCT *(in particular,* $\mathrm{OPT}_{\mathrm{FPT}}$ *is* $\mathrm{OPT}_{\mathrm{FCT}}$*).*

Proof. Note that $\mathrm{FPT}(A,\mathcal{F}) = \mathrm{FCT}(A,\mathcal{F}) + c(\mathcal{F})$, where $c(\mathcal{F})$ is a function that does not depend on A. Hence, $\mathrm{FPT}(A,\mathcal{F}) - \mathrm{FPT}(A',\mathcal{F}) = \mathrm{FCT}(A,\mathcal{F}) - \mathrm{FCT}(A',\mathcal{F})$. □

Theorem 2 shows that from a practical perspective, there is no difference between FPT and FCT since the minimization of FPT leads to the minimization of FCT and vice versa. In the following, we use FPT only in our theoretical analysis. In the rest of the paper, we denote by OPT the policy $\mathrm{OPT}_{\mathrm{FPT}}$ and by e_i^{OPT} the end time of f_i in S^{OPT}.

Note that mice flows are more latency-sensitive than elephant flows. Minimizing the value of FPT (and FCT) leads to latency reductions for mice flows [9]. However, due to different QoS requirements, latency sensitivity can differ even for flows of the same size. The heterogeneity of latency requirements can be incorporated into the final objective by assigning weights w_i to each flow $f_i \in \mathcal{F}$. In this case, we minimize the total *weighted flow processing time* WFPT defined as $W\mathrm{FPT}(A,\mathcal{F}) = \sum_{i=1}^{|\mathcal{F}|} w_i \cdot e_i^A$. In Sections V and VI, we will generalize proposed scheduling policies to the case of weighted flows preserving the competitiveness of these policies.

IV. Oblivious Online Scheduling Policies

The FIFO, BRR [20], LAS [18], and PIAS [6] policies scheduling flows in a switch S_1 with one ingress port are oblivious online since they do not exploit flow sizes. In this section, we study properties of these policies and provide their worst case latency performance guarantees in the general case.

At every time slot, FIFO transmits a packet that had arrived to S_1 first. The main disadvantage of FIFO is that packets of mice flows may be stuck on the ingress port since packets of elephant flows had arrived to S_1 earlier. For instance, in Fig. 1 FIFO cannot transmit packets of f_2 and f_3 before the end time of f_1. Theorem 3 shows that FIFO does not have a constant competitive factor w.r.t. to FPT.

Theorem 3. *For any $n > 0$, there exists a set of flows $\mathcal{F}$ such that* $\mathrm{FPT}(\mathrm{FIFO},\mathcal{F}) \geq \frac{n}{2} \cdot \mathrm{FPT}(\mathrm{OPT},\mathcal{F})$.

Proof. Consider the following set $\mathcal{F}$ of n flows: f_1 consists of $l \gg n^2$ packets arriving to S_1 in the first time slot; the other $(n-1)$ flows in $\mathcal{F}$ have one packet each and arrive to S_1 in the second time slot. FIFO transmits all packets in f_1 before the transmission of a packet in any other flow in $\mathcal{F}$. Hence, $\mathrm{FPT}(\mathrm{FIFO},\mathcal{F}) \geq ln$. OPT can transmit packets in all flows in $\mathcal{F}$ except f_1 before the second packet of f_1. Thus, $\mathrm{FPT}(\mathrm{OPT},\mathcal{F}) \leq l + cn^2$, where c is the some constant. Taking $l > cn^2$ we obtain that $\mathrm{FPT}(\mathrm{OPT},\mathcal{F}) < 2l$. □

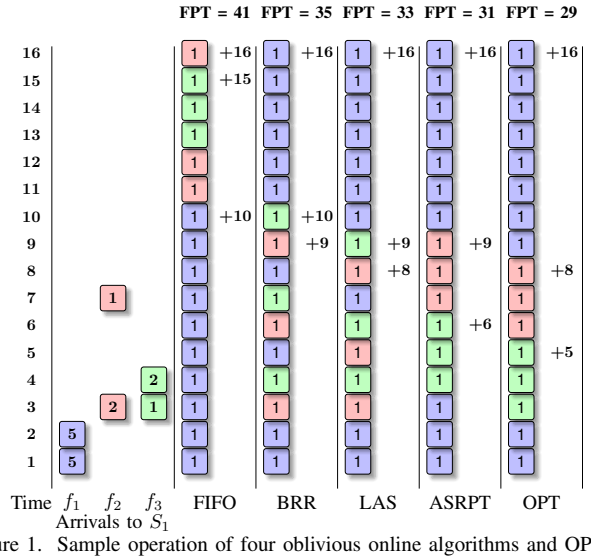

Figure 1. Sample operation of four oblivious online algorithms and OPT on three flows. Notation: packets from the same flow are shown in the same color; a rectangle marked n denotes a group of n packets from the considered flow; numbers in bold to the right of packets (such as $+10$ on time 10 for FIFO) show how much is added to FPT at this time.

To avoid the limitations of FIFO, we can apply *fair* scheduling policies BRR and LAS that distribute the transmission time of S_1 among all flows in $\mathcal{F}$ "evenly". In this case, packets from mice flows cannot get stuck on the ingress port of S_1.

BRR processes all flows having packets on the ingress port of S_1^{BRR} in a cycle; on every iteration of this cycle BRR transmits a single packet in the current flow. In Fig. 1, $\mathrm{FPT}(\mathrm{BRR},\mathcal{F})$ is significantly smaller than $\mathrm{FPT}(\mathrm{FIFO},\mathcal{F})$ since BRR evenly transmits packets from all flows between the third and tenth time slots. Unlike FIFO, the BRR policy has a constant competitive factor w.r.t. FPT.

Theorem 4. BRR *is at most* 3*-competitive w.r.t.* FPT.

Proof. Consider a fixed set of flows $\mathcal{F}$. We assume that flows in $\mathcal{F}$ are enumerated in the increasing order of their sizes, i.e., $l_1 \leq l_2 \leq \ldots \leq l_{|\mathcal{F}|}$ Let R_i be the set of all time slots when f_i has packets on the ingress port of the switch S_1^{BRR}. Note that for each flow $f_i \in \mathcal{F}$, $e_i^{\mathrm{BRR}} - |R_i|$ does not exceed the time when the last packet of f_i arrives to S_1 and hence does not exceed e_i^{OPT}. Therefore, to show 3-competitiveness of BRR it suffices to prove the following: $\sum_{i=1}^{|\mathcal{F}|} |R_i| \leq 2 \cdot \mathrm{FPT}(\mathrm{OPT},\mathcal{F})$.

Let $T_i = t_1, t_2, \ldots, t_{l_i}$ be a sequence of all time slots when BRR transmits a packet of the flow f_i. Note that $|T_i| = l_i$. Let t_{x-1} and t_x be a pair of consecutive time slots in T_i. By the definition of BRR, for each $j > i$ the set R_i contains at most one time slot t such that $t \in T_j$ and $t_{x-1} \leq t \leq t_x$ (for $x = 1$ we assume that $t_{x-1} = a_i$). Hence, $|R_i|$ can be bounded as follows: $|R_i| \leq (|\mathcal{F}| - i + 1) \cdot |T_i| + \sum_{j=1}^{i-1} l_j$, where $\sum_{j=1}^{i-1} l_j$ is an upper bound on the number of time slots when BRR transmits the packets of the first $i-1$ flows. Summing $|R_i|$ over all flows in $\mathcal{F}$ we obtain the following: $\sum_{i=1}^{|\mathcal{F}|} |R_i| \leq \sum_{i=1}^{|\mathcal{F}|} (2(|\mathcal{F}| - i) + 1) \cdot l_i$.

On the other hand, the value of $\sum_{i=1}^{|\mathcal{F}|} \sum_{j=1}^{i} l_j = \sum_{i=1}^{|\mathcal{F}|} (|\mathcal{F}| - i + 1) \cdot l_i$ is a lower bound on $\mathrm{FPT}(\mathrm{OPT},\mathcal{F})$

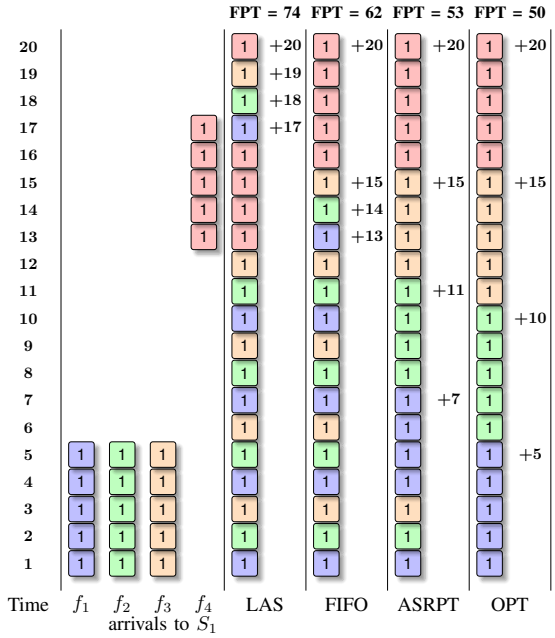

Figure 2. Sample operation of LAS, FIFO, ASRPT, and OPT on four flows; notation is explained in the caption of Fig. 1.

since all packets of the first i flows in $\mathcal{F}$ cannot be transmitted by any algorithm in time less than $\sum_{j=1}^{i} l_j$. □

Note that BRR transmits packets evenly between mice and elephant flows. To minimize FPT in practice, an efficient scheduling policy should prefer transmitting packets of mice flows over packets of elephant flows. This idea is reflected in the LAS scheduling policy [18]. At every time slot, LAS transmits a packet with the smallest packet number. For instance, in Fig. 1 LAS transmits the third packet of f_1 only after the transmission of the first two packets of f_2 and f_3.

Note that packet numbers of most packets in elephant flows exceed the sizes of mice flows. Hence, similar to SRPT, LAS transmits packets of mice flows first. LAS also has a constant competitive factor w.r.t. FPT.

Theorem 5. LAS *is at most* 3*-competitive w.r.t.* FPT.

Proof. We use the same notation as in Theorem 4, replacing BRR by LAS. Similar to Theorem 4, it suffices to show that $|R_i| \leq (|\mathcal{F}| - i + 1) \cdot l_i + \sum_{j=1}^{i-1} l_j$. Note that at every time slot in R_i, LAS transmits a packet with number at most l_i. Summing the number of such packets in all flows, we obtain the desired bound on R_i. □

Unfortunately, fair policies do not schedule elephant flows of similar sizes efficiently. For instance, consider a set $\mathcal{F}$ of four flows shown in Fig. 2. In this case, LAS has the largest FPT among all policies that transmit a packet on each of the first 20 time slots; even FIFO significantly outperforms LAS. Here LAS transmits 4 packets from flows f_1, f_2, f_3 during the first 12 time slots, i.e., after time slot 12 each of these three flows has only one packet left. Despite this, LAS finishes the transmission of the first three flows in $\mathcal{F}$ only after the transmission of four packets from f_4. In practice, such situations can lead to a huge starvation of elephant flows that concurrently arrive to S_1.

The scheduling policy in PIAS [6] is the combination of FIFO and LAS. PIAS maintains k thresholds $\alpha_1, \alpha_2, \ldots \alpha_k$ and defines the *relaxed packet number* of a packet p as the largest index of the threshold not exceeding the packet number of p. At every time slot, PIAS transmits a packet with the smallest relaxed packet number, breaking ties by FIFO. PIAS allows to use a fixed number of priorities in S and reduces the starvation of elephant flows. To minimize FCT, PIAS periodically changes the threshold values according to the network load and the flow size distribution. However, at least in the case when predicted thresholds mismatch arriving traffic patterns, PIAS does not have a constant competitive factor.

Theorem 6. *For any $n > 0$ and any fixed set of k thresholds $\alpha_1, \alpha_2, \ldots \alpha_k$, there exists a set of flows $\mathcal{F}$ such that* $\mathrm{FPT}(\mathrm{PIAS}, \mathcal{F}) \geq \frac{n}{2} \cdot \mathrm{FPT}(\mathrm{OPT}, \mathcal{F})$.

Proof. Multiplying the sizes of flows in the proof of Theorem 3 by α_k, we obtain that $\mathrm{FPT}(\mathrm{PIAS}, \mathcal{F}) \geq \frac{n}{2} \cdot \mathrm{FPT}(\mathrm{OPT}, \mathcal{F})$ since in this case, the relaxed packet number equals k for $\alpha_k(l-1)$ packets in f_1 and the last packets in all other flows. □

In the next section, we propose a pure online generalization of SRPT overcoming the limitations of the above policies.

V. SRPT-based scheduling with unknown flow sizes

Recall that the SRPT policy on a switch S_1 (with a single ingress port) always transmits a packet of a flow f with the shortest remaining processing time $rpt(f)$. Note that SRPT has the following two major drawbacks: (1) SRPT is not a oblivious online policy since it requires prior knowledge of flow sizes; (2) SRPT does not take into account gaps between arrival times of packets in the same flow; for instance, in Fig. 3 SRPT selects f_1 for transmission over f_2, while OPT chooses f_2 over f_1 since the last packet of f_1 arrives to S_1 significantly later than its first packet.

To overcome the drawbacks of SRPT, we propose an oblivious online algorithm *agnostic* SRPT (ASRPT) that splits a set of flows having packets on the ingress port of S_1 into two groups: (1) flows whose last packets have arrived to S_1; (2) all other flows. If the first group contains at least one flow, ASRPT transmits a packet from a flow in this group with the shortest remaining processing time. Otherwise, ASRPT transmits a packet from a flow in the second group in FIFO order. Note that ASRPT is oblivious online since it calculates $rpt(f)$ only after the arrival of the last f's packet to S_1.

In Fig. 3, similar to SRPT, ASRPT transmits packets from f_3 on the third and fourth time slots since f_3 is smaller than f_1 and f_2. However, unlike SRPT, ASRPT selects f_2 for transmission over f_1 after the first time slot since the last packet of f_2 arrives to S_1 at the second time slot. This example shows that ASRPT takes into account the differences between arrival times of packets in the same flow.

Typically, the last packet of an elephant flow arrives to S_1 significantly later than its first packet, i.e., an elephant flow belongs to the second group in ASRPT most of the time. Hence, separating flows into two groups also helps to select mice flows over elephant flows during a transmission phase.

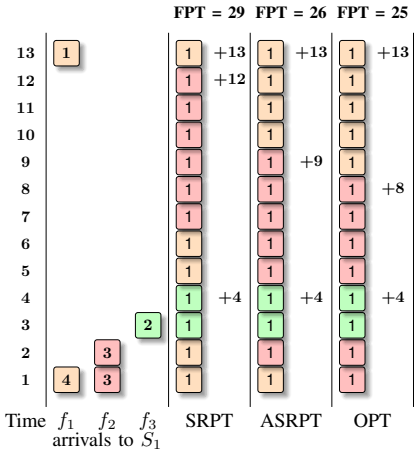

Figure 3. Sample operation of SRPT, ASRPT and OPT on three flows; notation is explained in the caption of Fig. 1.

Unlike LAS, ASRPT can schedule elephant flows of similar sizes efficiently. In Fig. 2, ASRPT obtains the sizes of f_1, f_2, and f_3 only at the fifth time slot. Despite such delay, ASRPT significantly outperforms LAS and FIFO. In the evaluation study, ASRPT significantly outperforms LAS on elephant flows. In Fig. 1, ASRPT also outperforms all other considered algorithms except OPT. Theorem 7 shows that theoretical guarantees improve as well (see Appendix for the proof).

Theorem 7. ASRPT *is at most* 2*-competitive w.r.t.* FPT.

Note that flows in the second group of ASRPT can also be scheduled by BRR, LAS or any other scheduling policy that does not exploit flow sizes. The competitiveness of these ASRPT implementations remains the same. Replacing $rpt(f_i)$ by $\frac{rpt(f_i)}{w_i}$ in ASRPT, we obtain an extension of ASRPT to the case of weighted flows, where w_i is the weight of f_i. This extension is at most 2-competitive w.r.t. to WFPT (the proof is the same as in Theorem 7).

VI. Removing dependency on flow sizes

So far we have considered policies scheduling flows in a switch S_1 with a single ingress port. Now we turn to the case of a switch S containing multiple ingress and egress ports. In the following, we propose a general method transforming a policy A on S exploiting flow sizes into the oblivious online policy GA. The transformation of A into GA is based on the same design principles as the transformation of SRPT into ASRPT (see Section V). GA is shown in Algorithm 1.

Similar to ASRPT, GA divides a set of flows with packets on their ingress ports into two groups: (1) flows whose last packets have been arrived to S; and (2) all other flows. GA schedules flows in the first group in the same way as A (line 3 in Algorithm 1) and flows in the second group according to the FIFO policy (line 7 in Algorithm 1). GA prefers transmitting packets from flows in the first group. Since selected flows cannot share the same ingress or egress port, GA does not consider flows in the second group sharing ports with flows chosen by A for transmission (line 5 in Algorithm 1). Theorem 8 shows that the competitiveness of A in the case of gapless flows (flows whose packets arrive to S without time gaps) translates into the competitiveness of GA in the general case with only a constant factor (see Appendix for the proof).

Algorithm 1 GA

1: $\mathcal{F} \leftarrow$ set of flows having packets on their ingress ports
2: $G_1 \leftarrow$ set of flows in $\mathcal{F}$ whose last packets have arrived to S
3: $\text{Result}_1 \leftarrow$ flows in G_1 that are chosen by A for transmission
4: $G_2 \leftarrow \mathcal{F} \setminus G_1$
5: $G_2 \leftarrow G_2 \setminus \{f_i : \text{in}_i = \text{in}_j \text{ or } \text{out}_i = \text{out}_j \text{ for some } f_j \in \text{Result}_1\}$
6: $\text{Result}_2 \leftarrow$ flows in G_2 that are chosen by FIFO for transmission
7: $\text{Result} \leftarrow \text{Result}_1 \cup \text{Result}_2$
8: transmit first untransmitted packets of the flows in Result

Theorem 8. *If A is k-competitive w.r.t.* FPT *in the case of gapless flows then* GA *is at most $3k$-competitive w.r.t.* FPT *in the general case.*

Applying the proposed above transformation to SRPT we obtain the ASRPT policy. Hence, Theorem 8 immediately implies that ASRPT is at most 3-competitive w.r.t. FPT. This bound on ASRPT competitiveness is weaker than in Theorem 7 since Theorem 8 shows properties of general scheduling policies managing a switch S that may contain multiple ingress and egress ports. Similarly to ASRPT, GA can also schedule flows in the second group by any scheduling policy that does not exploit flow sizes. The competitiveness of these GA implementations remains the same. In the case of weighted flows, GA is at most $3k$-competitive w.r.t. WFPT if the corresponding A is k-competitive w.r.t. WFPT (the proof is the same as in Theorem 8).

VII. *ASCC* design

SRPT-based congestion controls approximate the performance of the *Ideal* non-oblivious online policy on a switch S scheduling flows greedily according to remaining flow sizes (see Algorithm 1 in [5]). The transformation from Section VI can convert *Ideal* into an oblivious online policy *GIdeal*. This section shows how to incorporate the proposed SRPT generalizations into DCTCP [1] in order to approximate the performance of the *GIdeal* policy. We call the resulting congestion control *ASCC* and describe four general design principles of *ASCC*. Below, we say that a flow f is *complete* if its last packet is available for transmission at the source of f; otherwise, f is *incomplete*.

Buffer management policy. Each source implementing *ASCC* marks packets by a special bit that allows other network elements to determine whether the packet is from a complete or an incomplete flow. Each network element maintains separate queues for packets from complete and incomplete flows. Packets from complete flows are transmitted according to SRPT priorities specified by a flow's source, and packets from incomplete flows are transmitted according to FIFO. A network element transmits packets from incomplete flow only if there are no packets from complete flows. When the buffer overflows, the *ASCC* buffer management policy drops the last arriving packet from an incomplete flow; if there is no such packet, the lowest priority packet from a complete flow is dropped.

Retransmit when complete. Let us fix a flow f. Consider a set L of packets from f that are not acknowledged by f's

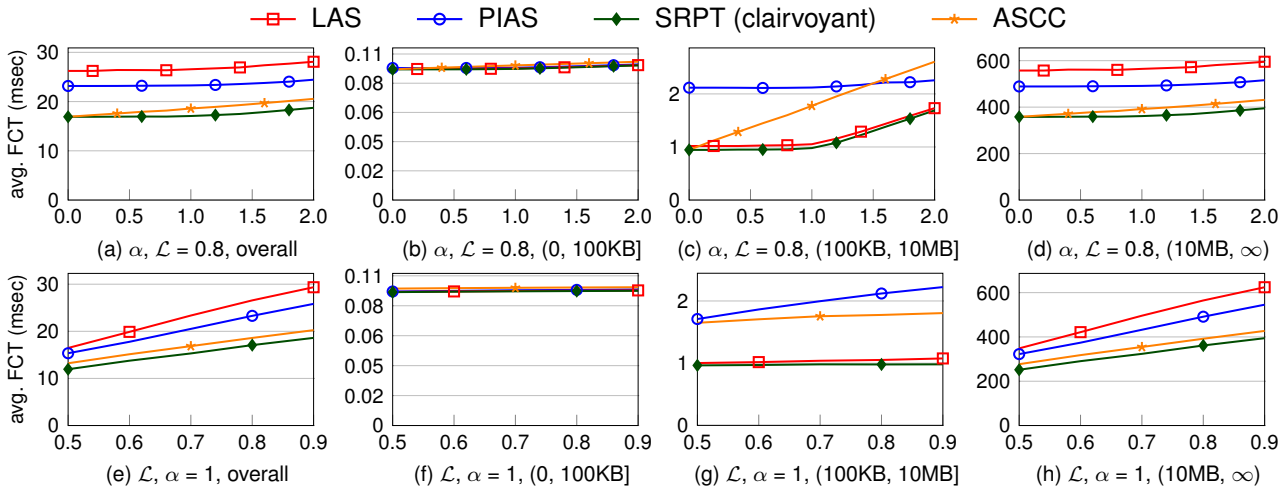

Figure 4. AFCT for *ASCC*, SRPT, PIAS, and LAS on the *data-mining* workload (**DM**) among: (a, e) all flows; (b, f) mice flows; (c, g) medium-size flows; (d, h) elephant flows. In (a-d) we vary α reflecting the spacing between arrival times of packets from the same flow; in (e-h) we vary the load $\mathcal{L}$.

destination D and were sent by f's source S before f became complete. Network elements transmit packets from L only if their buffers do not contain packets from complete flows. Hence, a packet from L can arrive to D with a noticeable delay or can be dropped with a much higher probability than packets from complete flows. Thus, when f becomes complete, *ASCC* transmits copies of packets in L marking them as for a complete flow. Note that packets in L can arrive to D later than these retransmitted copies, but D ignores all such packets.

Adjust `cwnd` when complete. The congestion window `cwnd` of incomplete flows can be very low since packets from complete flows are transmitted before packets from incomplete flows. Hence, when a flow becomes complete its congestion window should be increased. Before the transmission of packets from a flow that has just become complete, *ASCC* makes `cwnd` no less than the bandwidth-delay product in the idle network.

Ignore ECN from incomplete flows. In DCTCP, a flow's destination transmits ECN to a flow's source if for an incoming packet p, in at least one network element, the number of packets transmitted by this element between p's enque and p's deque exceeds a certain threshold. Due to the same reasons as in previous two cases, for *ASCC* packets from incomplete flows are much more likely to exceed the threshold. Once a flow becomes complete, ECNs generated for packets that are already in the network will not reflect the queuing delay the future packets will experience. Thus, after this point *ASCC* ignores all ECNs caused by packets sent before the flow became complete.

Observe that with the growth of the network load, the performance of *ASCC* w.r.t. FCT becomes close to the performance of corresponding SRPT-based congestion controls that know flow sizes in advance since flows become complete more frequently. At the same time, high network load can lead to significant performance degradation of LAS-based congestion controls due to the mutual starvation of elephant and medium-size flows.

Intuitively, *ASCC* gives to complete flows as much network resources as they need and transmits these flows according to SRPT. The remaining network resources are used to transmit incomplete flows. For instance, if flow fragments contain intermediate results of SQL queries, *ASCC* prefers transmitting results of queries whose execution has finished.

VIII. Experimental Evaluation

In this section we present a comprehensive evaluation of *ASCC*'s ability to optimize *average flow completion times (AFCT)*. The simulation code is publicly available at [21].

Topology. We use the same leaf-spine topology as *pFabric* [5] and PIAS [6] with 4 spine switches, 9 ToR leaf switches, and 16 servers connected to each ToR switch, i.e., 144 servers in total. Server-to-ToR link rates are 10Gbps, ToR-to-spine are 40Gbps. Every two servers exchange flows independently.

Workloads. Flow sizes are generated from empirical *web-search* [1] (**WS**) and *data-mining* [22] (**DM**) distributions that have been extensively used in the literature (see *pFabric* [5] and PIAS [6]). In the evaluation, the arrival time of a packet is the time when it becomes available to the network for transmission and the arrival time a_i of a flow f_i is the arrival time of the f_i's first packet. Following [5] and [6], we generate flow arrival times by $\text{Pois}(\lambda)$ distribution: for a given load $\mathcal{L} \in (0, 1)$ the flow arrival rate $\lambda(\mathcal{L})$ is derived as in PIAS [6]. Due to the lack of information on packet arrival times, we generate them synthetically. Let Δt_i be the minimum possible time that it takes to send all packets of a flow f_i out of the sender's host interface. We distribute arrival times of f_i's packets uniformly in $[a_i, a_i + \alpha\Delta t_i]$, where $\alpha \geq 0$ is a parameter reflecting the spacing between arrival times of f_i's packets.

Evaluated CCs. We compare *ASCC* with three other *congestion controls (CCs)*: SRPT and LAS [18] with DCTCP [1] end host control logic and PIAS [6]. LAS and PIAS do not rely on *a priori* known flow sizes, and we augment SRPT with a clairvoyant ability to learn the flow size at the moment when the first packet arrives to the flow source. We set buffer sizes bs and $\text{rto}_{\min}$ following PIAS's evaluations: $bs = 240$ pkts. and $\text{rto}_{\min} \approx 24\text{RTT}$. Note, all four CCs use DCTCP-based control logic at end hosts allowing for a fair comparison w.r.t. scheduling at the switches.

Performance metric. We compare AFCT while varying $\mathcal{L}$ from 0.5 to 0.9 and α from 0 to 2. As α grows from 0 to 1

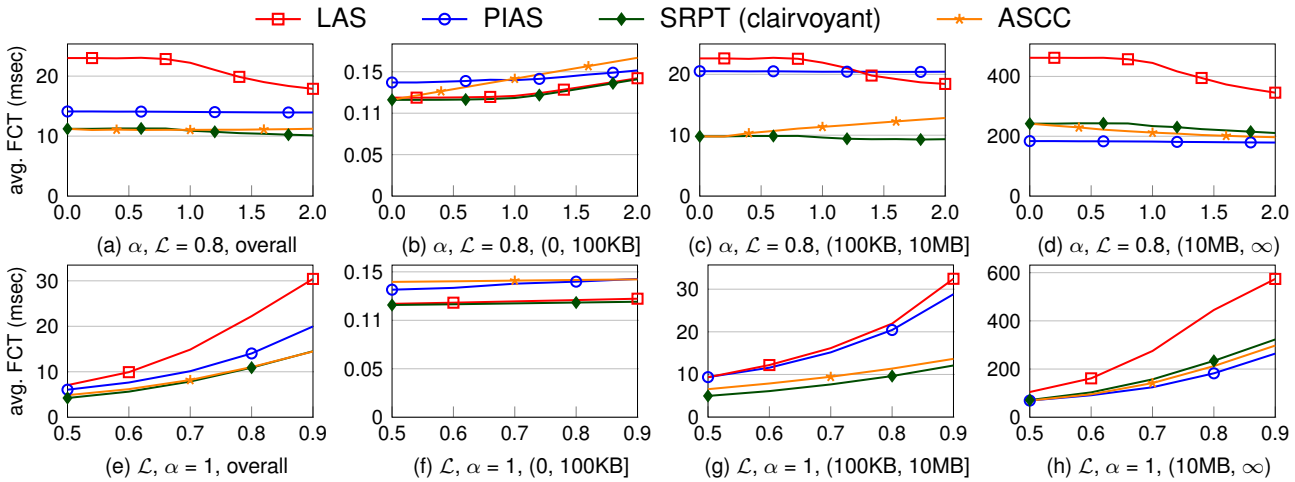

Figure 5. AFCT for *ASCC*, SRPT, PIAS, and LAS on the *web-search* workload (**WS**) among: (a, e) all flows; (b, f) mice flows; (c, g) medium-size flows; (d, h) elephant flows. In (a-d) we vary α reflecting the spacing between arrival times of packets from the same flow; in (e-h) we vary the load $\mathcal{L}$.

we expect PIAS, SRPT, and LAS not to change their behavior: first, flow packets arrive to the source faster than its interface transmits, so there are no gaps in flow data; second, none of the three use packet arrival times to set packet priorities. At the same time, *ASCC*'s AFCT grows slightly due to the true flow size being learnt later. As α grows further to 2.0, arrival times of flow packets become more and more dispersed in time, affecting AFCTs for all CCs. We show AFCT among all flows in Fig. 4(a,e) and Fig. 5(a,e), among mice flows (<100KB) in Fig. 4(b,f) and Fig. 5(b,f), among medium-size flows (100KB to 10MB) in Fig. 4(c,g) and Fig. 5(c,g), and, finally, among elephant flows (>10MB) in Fig. 4(d,h) and Fig. 5(d,h).

AFCT among mice flows. In **DM**, the average size of mice flows is 2.68 pkts with more than 60% having only one packet, for that reason, mice flows' AFCT mainly depends on FCTs of single-packet flows. All CCs transmit single-packet flows with near-optimal latency, so in Fig. 4(b,f) the relative difference in AFCT never exceeds 3%.

The average size of mice flows in **WS** is 21.77 pkts, that is, much larger than in **DM**. As a result, we see more variability in AFCT among the considered CCs. SRPT and LAS provide near-optimal latency for mice flows since both tend to transmit packets from a smaller flow before transmitting from substantially larger ones. In contrast, PIAS and *ASCC* much more frequently send a packet from a smaller flow *after* sending from a larger one, which leads to their lower mice-flow performance compared to LAS and SRPT (see Fig. 5(b,f)).

Focusing further on *ASCC vs* PIAS performance, in Fig. 5(b) at $\alpha = 0$, we see *ASCC* behaving exactly as SRPT with AFCT 18% less than PIAS's. As α increases from 0 to 2, the delay before *ASCC* learns the true flow size increases and manifests as higher AFCT. In particular, for $\alpha = 1$ the AFCT of *ASCC* begins to exceed that of PIAS and for $\alpha = 2$ it is 9% higher. In Fig. 5(f), where we set $\alpha = 1$, PIAS slightly outperforms *ASCC*, but the difference almost disappears at higher loads. *The above observations let us conclude that for mice flows* ASCC *is better than* PIAS *at small* α*s and higher loads.*

AFCTs of LAS and SRPT grow when α increases from 1 to 2 in Fig. 5(b) as data becomes available later in time. Over this interval *ASCC* and SRPT show exactly the same rate of AFCT growth, implying that *out of the two main reasons for* ASCC*'s AFCT increase, namely lack of knowledge and lack of data, the latter is the dominant factor.*

AFCT among medium-size flows. For medium-size flows from **DM** SRPT and LAS continue to show near-optimal behavior (see Fig. 4(c,g)). In Fig. 4(c) *ASCC* outperforms PIAS in a majority of cases (except $\alpha \geq 1.6$). In Fig. 4(g) *ASCC* is everywhere better than PIAS: *ASCC*'s AFCT is 10% lower at $\mathcal{L} = 0.5$ and is 33% lower at $\mathcal{L} = 0.9$. Also, when in Fig. 4(c) $\alpha \in [1, 2]$, PIAS's rate of AFCT growth is much lower relative to *ASCC*'s as PIAS suffers from its scheduling inefficiency, while *ASCC*'s growth is due to delayed data.

Compared to **DM** there are 3.3× more medium-size flows in **WS** with average size 2.5× larger. These two factors lead to an order of magnitude higher AFCTs for all four CCs (see Fig. 5(c,g)). In such a harsh environment, performance of PIAS and LAS degrades substantially due to mutual starvation among concurrent medium-size flows; e.g., at $\mathcal{L} = 0.9$ their AFCTs are, respectively, 2× and 2.3× higher than *ASCC*'s. In contrast, for *ASCC* a more contentious environment with higher FCTs means shorter, relative to FCTs, periods of uncertainty, which brings it closer to SRPT, and we see in Fig. 5(g) that *ASCC*'s AFCT never exceeds SRPT's by more than 10%.

Finally, in Fig. 5(c) AFCTs of SRPT and LAS start to decrease for $\alpha > 1$. Such a counter-intuitive effect is also seen on elephant flows, so we explain it in the next paragraph.

AFCT among elephant flows. LAS loses substantially to other CCs in both workloads (see Fig. 5(d,h) and Fig. 4(d,h)) by starving larger flows. SRPT and *ASCC* are almost identical for the same reasons as for medium-size flows in **WS**. PIAS's AFCT is up to 25% higher than SRPT's and *ASCC*'s for **DM** and 10% lower for **WS**. In the latter case, PIAS has a better AFCT for elephant flows only because it prefers packets from elephant flows more often than SPRT and *ASCC*. To verify this hypothesis, we ran all three CCs on elephant flows only, and indeed, both *ASCC* and SRPT outperformed PIAS.

The counterintuitive effect we have seen on medium-sized flows, namely decreasing AFCT with increasing α, becomes

very pronounced for LAS on elephant flows. There are two contributing factors. First, LAS has poor AFCT performance for similarly-sized flows, and higher α restricts this natural tendency of LAS. Second, larger α makes data arrival more evenly spread, reducing instantaneous load on the network and hence, the number of packet drops. SRPT exhibits the same effect but to a much lesser extent thanks to SRPT's near-optimal behavior.

AFCT among all flows. For both workloads, among the four approaches LAS shows the worst performance. The *ASCC*'s overall performance follows closely the near-optimal behavior of SRPT. Our *ASCC* approach outperforms LAS and PIAS in all experiments with up to 30% less AFCT in **`WS`** and with up to 25% less AFCT in **`DM`**.

The evaluation confirms that *ASCC* robustly deals with unknown flow sizes avoiding starvation of concurrent elephant and medium-size flows. In particular, as burst intensity increases, LAS-based CCs show severe performance degradation, while *ASCC*'s behavior mimics near-optimal clairvoyant SRPT.

IX. Related Work

Existing approaches that have been proposed for datacenter transports include both protocols that require knowledge of flow sizes and protocols that do not.

Requiring flow sizes. *pFabric* [5] is an information-aware transport mechanism that processes packets according to SRPT. *Fastpath* [23] allows to exploit commodity network fabrics with a centralized scheduler, but it loses many performance benefits provided by *pFabric*. *pHost* [8] attempts to achieve the performance of *pFabric* by running additional in-network control that lets it run on commodity switches with FIFO packet processing. *HyLine* [24] performs joint load balancing and path-aware flow scheduling, looking for a balance between centralized and distributed techniques. *Sincronia* [14] schedules coflows, minimizing coflow completion time. *Homa* [7] is a receiver-driven SRPT-based transport protocol expoloiting a fixed number of different priorities in the network elements. SRPT has also been studied under stochastic assumptions for task scheduling in multi-server environments [25].

Not requiring flow sizes. A number of recent designs use flow rate control to achieve desired performance characteristics. In particular, DCTCP, a flow size agnostic solution, uses an adaptive congestion control mechanism based on ECN to keep queue sizes small [1]. D2TCP [2] uses rate control to maximize the number of flows satisfying their respective deadlines, and MCP [26] enhances D2TCP with ECN. However, these transports do not support flow preemption and do not proritize packet processing as *pFabric* does [5], which can negatively affect performance. LAS [18] is flow size agnostic and uses transmitted flow sizes as priorities for the flows. PIAS [6] implements a LAS variation designed for commodity switches [6] and is based on the DCTCP. PDQ [4] and D3 [3] exploit network-wide arbitration but can introduce high flow switching overhead. [27] uses machine learning to predict flow sizes. PASE [28] is a combined solution that uses arbitration control plane, endpoint transport protocol that is aware of priority queues, and an adjustable rate control mechanism. [29] propose efficient load balancing flowlet schemes.

X. Conclusion

In this work, we study and analyze analytically congestion control mechanisms without prior knowledge of flow sizes. SRPT-based policies are very attractive with respect to performance, but by default they require flow sizes to be available a *priori*. We have proposed SRPT generalizations for this harsh environment. Extensive evaluations that we have carried out show that proposed SRPT generalizations are not only interesting from a theoretical perspective but also represent a viable practical solution at least on the considered workloads. We believe that other SRPT-based transports such as [7, 8] can incorporate the proposed design principles as well, and they can be tested by the commercial community under operational conditions, moving towards real deployment scenarios.

References

[1] M. Alizadeh *et al.*, "Data center TCP (DCTCP)," in *SIGCOMM*, 2010, pp. 63–74.

[2] B. Vamanan *et al.*, "Deadline-aware datacenter tcp (D2TCP)," in *SIGCOMM*, 2012, pp. 115–126.

[3] C. Wilson *et al.*, "Better never than late: meeting deadlines in datacenter networks," in *SIGCOMM*, 2011, pp. 50–61.

[4] C. Hong *et al.*, "Finishing flows quickly with preemptive scheduling," in *SIGCOMM*, 2012, pp. 127–138.

[5] M. Alizadeh *et al.*, "pfabric: Minimal near-optimal datacenter transport," in *SIGCOMM*, 2013, p. 435–446.

[6] W. Bai *et al.*, "Pias: Practical information-agnostic flow scheduling for commodity data centers," *IEEE/ACM Transactions on Networking*, vol. 25, no. 4, pp. 1954–1967, 2017.

[7] B. Montazeri *et al.*, "Homa: A receiver-driven low-latency transport protocol using network priorities," in *SIGCOMM*, 2018, pp. 221–235.

[8] P. X. Gao *et al.*, "phost: Distributed near-optimal datacenter transport over commodity network fabric," in *CoNEXT*, 2015, pp. 1:1–1:12.

[9] N. Dukkipati *et al.*, "Why flow-completion time is the right metric for congestion control," *SIGCOMM Comput. Commun. Rev.*, vol. 36, no. 1, pp. 59–62, Jan. 2006.

[10] A. Munir *et al.*, "Friends, not foes: synthesizing existing transport strategies for data center networks," in *SIGCOMM*, 2014, pp. 491–502.

[11] A. Mushtaq *et al.*, "Datacenter congestion control: identifying what is essential and making it practical," *SIGCOMM Comput. Commun. Rev.*, vol. 49, no. 3, pp. 32–38, 2019.

[12] A. Sivaraman *et al.*, "An experimental study of the learnability of congestion control," in *SIGCOMM*, 2014, pp. 479–490.

[13] K. Winstein *et al.*, "TCP ex machina: computer-generated congestion control," in *SIGCOMM*, 2013, pp. 123–134.

[14] S. Agarwal *et al.*, "Sincronia: Near-optimal network design for coflows," in *SIGCOMM*, 2018, pp. 16–29.

[15] R. T. Fielding *et al.*, "Hypertext transfer protocol - HTTP/1.1," *RFC*, vol. 2616, pp. 1–176, 1999.

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

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

[18] I. A. Rai *et al.*, "Performance analysis of las-based scheduling disciplines in a packet switched network," *SIGMETRICS Performance Evaluation Review*, vol. 32, pp. 106–117, 2004.

[19] A. Borodin *et al.*, *Online computation and competitive analysis*. Cambridge University Press, 1998.

[20] A. Greenberg *et al.*, "How fair is fair queuing?" *Journal of the ACM (JACM)*, vol. 39, pp. 568–598, 1992.

[21] "Srpt-based congestion control for flows with unknown sizes," https://github.com/pschuprikov/ascc.

[22] A. Greenberg *et al.*, "Vl2: A scalable and flexible data center network," in *SIGCOMM*, 2009.

[23] J. Perry *et al.*, "Fastpass: a centralized "zero-queue" datacenter network," in *SIGCOMM*, 2014, pp. 307–318.

[24] S. Abbasloo *et al.*, "Hyline: a simple and practical flow scheduling for commodity datacenters," in *IFIP*, 2018, pp. 334–342.
[25] I. Grosof *et al.*, "Srpt for multiserver systems," *SIGMETRICS Perform. Eval. Rev.*, vol. 46, no. 3, p. 8–9, Jan. 2019. [Online]. Available: https://doi.org/10.1145/3308897.3308902
[26] L. Chen *et al.*, "Towards minimal-delay deadline-driven data center TCP," in *HotNets*, 2013, pp. 21:1–21:7.
[27] V. Dukić *et al.*, "Is advance knowledge of flow sizes a plausible assumption?" in *NSDI*, 2019, pp. 565–580.
[28] A. Munir *et al.*, "PASE: synthesizing existing transport strategies for near-optimal data center transport," *IEEE/ACM Trans. Netw.*, vol. 25, no. 1, pp. 320–334, 2017.
[29] E. Vanini *et al.*, "Let it flow: Resilient asymmetric load balancing with flowlet switching," in *NSDI*, 2017, pp. 407–420.

APPENDIX

Here, we present proofs of Theorems 1, 7 and 8. The proof of Theorem 1 is based on Lemma 9. In the following, we denote by $active(A, \mathcal{F}, t)$ the number of active flows in S_1^A at time t, where $\mathcal{F}$ is the set of flows arriving to S_1.

Lemma 9. *Assume that algorithms A, A' satisfy the following: $active(A, \mathcal{F}, t) = l$ and $active(A', \mathcal{F}, t) = k$ for some $\mathcal{F}$ and t; then, there is a set of flows $\mathcal{F}'$ such that* $\mathrm{FCT}(A, \mathcal{F}') \geq \frac{l+1}{k+2} \cdot \mathrm{FCT}(\mathrm{OPT}, \mathcal{F}')$.

Proof. We construct $\mathcal{F}'$ by $\mathcal{F}$ in the following way: (1) we increase the arrival time of each flow packet in $\mathcal{F}$ arriving to S_1 at time t or later by a constant T exceeding $\mathrm{FCT}(A', \mathcal{F})$; (2) then, for each $0 \leq i < T$, we add to $\mathcal{F}'$ a flow that consists of a single packet and arrives to S_1 at time $t+i$. In the case of $\mathcal{F}'$, the switch S^A has at least $l+1$ active flows at every time slot t_1 such that $t \leq t_1 < t+T$. Observe that $\mathrm{FCT}(A, \mathcal{F}') = \sum_t active(A, \mathcal{F}', t)$. Hence, $\mathrm{FCT}(A, \mathcal{F}') \geq T \cdot (l+1)$.

Let A'' be the scheduling policy operating on $\mathcal{F}'$ at time t' as follows: (1) if $t' < t$ then A'' transmits the same packet as A' on $\mathcal{F}$ at time t'; (2) if $t \leq t' < T$ then A' transmits a packet in a flow arriving to S_1 at the current time slot; (3) otherwise, A' transmits the same packet as A' on $\mathcal{F}$ at time $t' - T$. $S^{A''}$ has exactly $k+1$ active flows at every time slot t_1 such that $t \leq t_1 < t+T$. Hence, $\mathrm{FCT}(A'', \mathcal{F}') = T \cdot (k+1) + \mathrm{FCT}(A', \mathcal{F})$. Since $T > \mathrm{FCT}(A', \mathcal{F})$, we obtain that $\mathrm{FCT}(A, \mathcal{F}') \geq \frac{l+1}{k+2} \cdot \mathrm{FCT}(A'', \mathcal{F}') \geq \frac{l+1}{k+2} \cdot \mathrm{FCT}(\mathrm{OPT}, \mathcal{F}')$ □

Proof of Theorem 1. Consider a scheduling policy A and a fixed set $\mathcal{F}$ of n flows consisting of l packets each. To prove the theorem, we define arrival times of flow packets in $\mathcal{F}$ and propose an algorithm A' such that $active(A, \mathcal{F}, t+1) = n$ and $active(A', \mathcal{F}, t+1) = 1$, where $t = l \cdot (n-1)$. In this case, the desired lower bound on the competitiveness of A directly follows from Lemma 9. In the following we assume that l can be divided by n^2+1 at least $2n$ times i.e., $l = c \cdot (n^2+1)^{2n}$.

First $l_1 = l \cdot (1 - \frac{1}{n^2+1})$ packets in each flow in $\mathcal{F}$ arrive to S_1 at the first time slot. There is at least one flow in $\mathcal{F}$ such that A transmits at most $l_1 \cdot (1 - \frac{1}{n})$ packets from this flow in the first $t_1 = l_1 \cdot (n-1)$ time slots; w.l.o.g. we assume that f_n is a such flow. The remaining $r_n = l - l_1$ packets of f_n arrive to S_1 at time $t_1 + 1$. Note that f_n will be active in S_1^A at time $t+1$ since the value of $t - t_1 + 1$ is no smaller than the number of packets in f_n that are not transmitted by A in the first t_1 time slots.

Next $l_2 = (l - l_1) \cdot (1 - \frac{1}{n^2+1})$ packets in each of first $n-1$ flows in $\mathcal{F}$ arrive to S_1 at time $t_1 + r_n + 1$. Again, there is at least one flow among first $n-1$ flows in $\mathcal{F}$ such that A transmits at most $l_1 + l_2 \cdot (1 - \frac{1}{n-1})$ packets from this flow in the first $t_2 = l_2 \cdot (n-2) + t_1 + r_n$ time slots; w.l.o.g. we assume that f_{n-1} is a such flow. The remaining $r_{n-1} = l - l_1 - l_2$ packets of f_{n-1} arrive to S_1 at time $t_2 + 1$. Similarly to f_n, the flow f_{n-1} will be active in S_1^A at time $t+1$ since the value of $t - t_2 + 1$ is no smaller than the number of packets in f_{n-1} that are not transmitted by A in the first t_2 time slots.

We recursively repeat the described above construction until only one flow remains; the last packet of f_1 arrives to S_1 at time $t+1$. As a result, all flows in $\mathcal{F}$ are active in S^A at time $t+1$. On the other hand, there is an algorithm A' transmitting all packets from all $\mathcal{F}$ flows except f_1 in the first t time slots, i.e., $active(A', \mathcal{F}, t+1) = 1$. □

Proof of Theorem 8. Consider a fixed set of flows $\mathcal{F} = \{f_1, f_2, \ldots, f_{|\mathcal{F}|}\}$. Assume that flows in $\mathcal{F}$ are ordered in the increasing order of the end time e_i^{OPT} in S^{OPT}. Let $\mathcal{F}' = \{f_1', f_2', \ldots, f_{|\mathcal{F}|}'\}$ be a set of gapless flows such that: (1) the arrival time a_i' of f_i' equals to the arrival time of the last packet of f_i; (2) the size l_i' of f_i' equals to the size l_i of f_i minus the number of packets in f_i transmitted by A before the time a_i'. By definition of A, $\mathrm{FPT}(\mathrm{GA}, \mathcal{F}) = \mathrm{FPT}(A, \mathcal{F}')$. Hence, $\mathrm{FPT}(\mathrm{GA}, \mathcal{F}) = \mathrm{FPT}(A, \mathcal{F}') \leq k \cdot \mathrm{FPT}(\mathrm{OPT}, \mathcal{F}')$.

Let ID be a policy that greedily selects flows in $\mathcal{F}'$ for the transmission according to the flow numbers. Since $\mathrm{FPT}(\mathrm{OPT}, \mathcal{F}') < \mathrm{FPT}(\mathrm{ID}, \mathcal{F}')$, to prove the theorem it is sufficient to show that $\mathrm{FPT}(\mathrm{ID}, \mathcal{F}') \leq 3 \cdot \mathrm{FPT}(\mathrm{OPT}, \mathcal{F})$. Let pin_i be the set of all packets in all flows $f_j' \in \mathcal{F}'$ such that $j < i$ and $\mathrm{in}_j = \mathrm{in}_i$. Similarly, pout_j is the set of all packets in all flows $f_j' \in \mathcal{F}'$ such that $j < i$ and $\mathrm{out}_j = \mathrm{out}_i$. The ID policy does not transmit a packet from a flow f_i' only if f_i' has not arrived yet or ID transmits a packet from the set $\mathrm{pin}_i \cup \mathrm{pout}_i$. Hence, the end time $\tilde{e}_i'$ of f_i' in S^{ID} does not exceed $a_i' + l_i' - 1 + |\mathrm{pin}_i| + |\mathrm{pout}_i|$,

Recall that a_i' also equals the arrival time of the last packet in a flow f_i, and flows in $\mathcal{F}$ are ordered in the increasing order of e_i^{OPT}. The end time e_i^{OPT} of f_i in S^{OPT} is not less than $\max(a_i', |\mathrm{pin}_i|, |\mathrm{pout}_i|) + l_i - 1$ since OPT transmits the last packet of f_i no earlier than any other packet in the first i flows of $\mathcal{F}$. As a result, $\tilde{e}_i' \leq 3 \cdot e_i^{\mathrm{OPT}}$ since $l_i' \leq l_i$ and $a + b + c \leq 3 \cdot \max(a, b, c)$ for any a, b, and c. Summing over all i from 1 to $|\mathcal{F}|$ we obtain that $\mathrm{FPT}(\mathrm{ID}, \mathcal{F}') \leq 3 \cdot \mathrm{FPT}(\mathrm{OPT}, \mathcal{F})$ □

Proof of Theorem 7. Recall that SRPT is OPT for gapless flows on S_1. Therefore, it is sufficient to prove that for S_1, in Theorem 8, the competitiveness of GA w.r.t FPT in the general case is only two times bigger than the competitiveness of A w.r.t. FPT in the case of gapless flows. Hence, it is sufficient to show that in the proof of Theorem 8, $\mathrm{FPT}(\mathrm{ID}, \mathcal{F}') \leq 2 \cdot \mathrm{FPT}(\mathrm{OPT}, \mathcal{F})$ if the considered switch contains a single ingress port. In this case, $\mathrm{pin}_i = \mathrm{pout}_i = \mathrm{pin}_i \cup \mathrm{pout}_i$. Hence, $\tilde{e}_i' \leq a_i' + l_i' - 1 + |\mathrm{pin}_i| \leq 2 \cdot \max(a_i', |\mathrm{pin}_i|, |\mathrm{pout}_i|) + l_i - 1 \leq 2e_i^{\mathrm{OPT}}$ implying that $\mathrm{FPT}(\mathrm{ID}, \mathcal{F}') \leq 2 \cdot \mathrm{FPT}(\mathrm{OPT}, \mathcal{F})$. □