2016 IEEE 36th International Conference on Distributed Computing Systems

Backlog-Aware SRPT Flow Scheduling in Data Center Networks

Tong Zhang, Fengyuan Ren, Ran Shu
Tsinghua National Laboratory for Information Science and Technology, Beijing, China
Dept. of Computer Science and Technology, Tsinghua Univeresity, Beijing, China
{zhangt, renfy, shuran}@csnet1.cs.tsinghua.edu.cn

***Abstract*—The rapidly developing soft real-time data center applications impose stringent delay requirements on internal data transfers. Therefore many recently emerged network protocols in data center share a common goal of decreasing *Flow Completion Time* (FCT), in which case the *Shortest Remaining Processing Time* (SRPT) scheduling discipline has attracted widespread attentions. However, SRPT suffers the instability issue, incurring more and more flows left uncompleted even when traffic load is within network capacity, which implies unnecessary bandwidth waste. To solve the problem, this paper proposes a backlog-aware scheduling algorithm (BASRPT) that stabilizes queue length while maintaining relatively low FCT based on Lyapunov optimization. To overcome the huge computational overhead, a fast and practical approximation algorithm called fast BASRPT is also developed. Extensive flow-level simulations show that fast BASRPT indeed stabilizes switch queue and obtains a higher throughput while being able to push FCT arbitrarily close to the optimal value in the condition of feasible traffic load.**

***Index Terms*—Backlog-Aware, SRPT, Flow Scheduling, Data Center Network**

I. Introduction

In today's data centers, soft real time applications impose stringent requirements on delay for short flows, in which case many recently proposed data center transport protocols share a common goal of minimizing *Flow Completion Times* (FCT) [1–5]. Therefore, the *Shortest Remaining Processing Time* (SRPT) scheduling algorithm, which has been theoretically proved to minimize the mean response time (sample path) [6], has received extensive attentions in data center transport designs. Assuming flow sizes are known a priori, PDQ [2], pFabric [3], and PASE [4] all adopt the idea of SRPT to effectively reduce the average FCT.

SRPT is a preemptive size-based flow scheduling algorithm, similar to the *Shortest Job First* (SJF) in the job scheduling issue of computing systems. The main idea is to always select the flow with the least remaining time until completion to run. Not only proved in theory, the delay performance of SRPT has also been evaluated with experiments in [7].

However, literatures have demonstrated SRPT reduces the stability region in linear networks [8–10]. In data center fabrics, the stability of the SRPT scheduling algorithm has only received limited attentions so far. Experiments in [11] discover the instability phenomenon, whereas no solution is provided. Here, stability is in the stochastic sense, meaning the recurrence of queue evolution process, and instability thus implies there could be more and more uncompleted flows left in queues. If this phenomenon occurs when traffic loads are less than network capacity, the bandwidth would be directly wasted, and the overall throughput will then be harmed. Therefore, stability is a crucial property in flow scheduling issues, and should be carefully studied.

We demonstrate the instability property of SRPT using a simple example that could really appear in data center flow patterns. And a subsequent simulation confirms the instability phenomenon in the data center fabric.

In order to understand the flow scheduling in data centers, we start by abstracting the whole data center fabric into a non-blocking input queued switch, as many state-of-art data center transport protocols did [3, 12–15]. By this abstraction, we get the expression to describe the queue length evolution process.

Depending on the above insights, we develop BASRPT, a backlog-aware flow scheduling algorithm that takes both queue lengths and remaining flow sizes into account. Each time making a scheduling decision, BASRPT scheduler traverses all possible scheduling schemes and chooses the one with the smallest $V \times average\ size\ of\ selected\ flows - total\ length\ of\ selected\ queues$, where $V \geq 0$ is an importance weight on FCT minimization against queue length stabilization. The basic idea of BASRPT follows Lyapunov optimization theory, by means of which BASRPT inherits the delay performance of SRPT and meanwhile stabilizes the switch queue.

However, BASRPT needs to traverse all possible scheduling schemes on each update, and each scheme contains massive computing, resulting in huge total computational overheads. Besides, the lack of explicit priorities among flows makes no way for any distributed implementation for BASRPT. Since scheduling decision updates on every arrival and completion whose occurring is rather frequent, the huge overhead makes BASRPT impractical to normally work in real systems. Thus we propose fast BASRPT, the heuristic approximation of BASRPT, which only involves the acceptable computational overheads. In data center fabrics, fast BASRPT selects flows one by one. When constructing a scheduling decision, each time the scheduler chooses the flow with the smallest $\frac{V}{N} \times flow\ size - queue\ length$, where N represents the total number of servers. Adding up the values of N chosen flows exactly produces $V \times average\ flows\ size - total\ selected\ queue\ length$. In this way, fast BASRPT approximately achieves the optimization goal of BASRPT. Since fast BASRPT assigns global

1063-6927/16 $31.00 © 2016 IEEE
DOI 10.1109/ICDCS.2016.26

priorities to all flows, it can be simply implemented using distributed paradigms [3].

We evaluate our algorithm using simulations on the traffic pattern obtained from existing measurements [1, 16]. We also discuss the influence of parameter V on the fast BASRPT performance. Simulation results reveal that fast BASRPT indeed stabilizes queue lengths at high loads while keeping the FCTs within the range of not affecting the application performance. When the traffic load is relatively low, BASRPT achieves almost identical delay performance as SRPT. Besides, the throughput is also improved in fast BASRPT. The parameter V properly makes a tradeoff between delay and stability, following *Lyapunov Optimization Theorem* [17].

In summary, the contributions of this work are as follows:

- Introducing the stability analysis as well as Lyapunov optimization to flow scheduling issues in data center networks;
- Designing a backlog-aware flow scheduling algorithm based on Lyapunov optimization, and proving its stability and deducing delay bound;
- Developing a practical and fast heuristic algorithm to approximates the exact algorithm;
- A systematic evaluation of the proposed algorithm over realistic traffic patterns and a wide range of key parameter

The rest of the paper is organized as follows. In Section II we introduce the SRPT scheduling algorithm as well as the related work on its stability property. We also illustrate its instability problem in the data center fabric using a simple example. In Section III, we establish the model for queue length evolution process. Based on *Lyapunov Optimization Theorem*, a backlog-aware flow scheduling algorithm (BASRPT) and its stability proof are proposed in Section IV. Besides, we develop a practical and fast version of BASRPT. We evaluate the proposed algorithm in Section V . In the end, Section VI concludes the paper.

II. Background and Motivation

In this section, we mainly introduce the SRPT scheduling algorithm employed by transport protocols in data centers as well as its stability investigations up to date. Further, through a simple example, we illustrate the instability of SRPT in the context of flow scheduling issue, which serves as the motivation of this work.

A. Background

In recent years, soft real-time data center applications have gained rapid development, such as web search, retail, advertisement, social networking, and recommendation systems [1]. A soft real-time application aggregates responses from many back-end servers to produce results. The tardy back-end flows will either inflate the application response time or be directly left out, so as to lower the quality of final results [2]. Since a small network delay will largely affect the user experience, these applications impose demanding delay requirements on the internal data transfers in data center networks.

Motivated by delay requirements, many recently proposed data center transport protocols share a common goal of minimizing FCT. In this case, the SRPT scheduling algorithm is widely used, since it has been theoretically proved to minimize the average FCT over a single link [6]. For example, PDQ [2], pFabric [3], and PASE [4] all adopt the basic idea of SRPT.

SRPT is a preemptive version of SJF, a size-based scheduling algorithm, in which the size of all existing processes (e.g., flows) must be known a priori. In the community of data center network protocol designs, many existing solutions assume accurate flow information, including known flow sizes [2–5, 12], in which case SRPT really applies. When scheduling over one single link, SRPT always selects the flow with the smallest remaining time until completion. It is the preference to flow with small size that minimizes the waiting time of arrived flows, namely, minimizing the average FCT. However, when simultaneously scheduling multiple links, the problem of minimizing average FCT is equivalent to the NP-hard *sum-multicoloring problem* even under non-blocking fabric assumption [18]. As a consequence, some recent data center transports adopt a simple approximate greedy algorithm, which schedules flows in a non-decreasing order of the remaining flow size until all left flows are blocked by currently selected ones [2–4]. From the algorithm itself, we can see that it is also SRPT. Fortunately, the approximate algorithm is theoretically guaranteed to provide near-ideal performance [18].

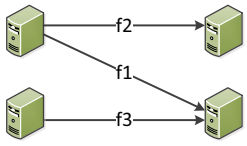
However, at the same time of the guaranteed delay performance, the stability properties of SRPT have been questioned more than once [8, 10, 11]. The stability here represents recurrence of the queue evolution process, meaning that any queue state seen before will always appear again. That is, there cannot be conditions in which a queue keeps growing and never comes back. With an unstable scheduling algorithm, there could be more and more uncompleted flows left even when the traffic load is in link capacity. In this context, the scheduling algorithm fails to use the available bandwidth efficiently and thus not yield optimal throughput performance.

There have been existing works demonstrating SRPT's instability in multi-link systems. The literature [8] establishes the exact stability conditions for SRPT in linear networks, and indicates SRPT may cause instability even at arbitrarily low traffic loads. The scenarios where SRPT reduces the stability region are illustrated in [10] using 2 independent M/D/1 queues. Furthermore, the authors prove that using SRPT locally within each traffic class is sure to improve the delay performance while maintaining the stability of networks.

However, the stability problem of SRPT in data center fabrics has not received enough attentions so far, and the few solutions on locally adopting SRPT eliminates the potential benefits of global SRPT scheduling. This provides the design space for a stable but delay-outstanding scheduling algorithm.

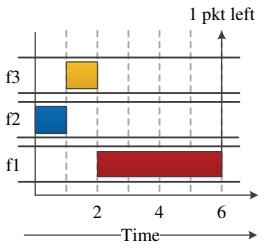
B. Motivation

The poor stability properties of SRPT in linear networks has been fully studied. Drawing upon its insights, we illustrate the instability that could happen in data center networks through

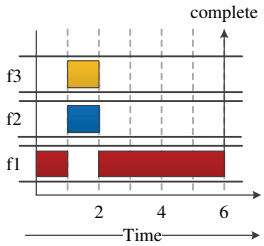


Flow	f1	f2	f3
Arrival Time	0	0	1

(a) Flow interaction and arrival times



(b) SRPT



(c) Backlog-aware

Fig. 1: An example of SRPT instability

a simple example: Consider a simple case of the scheduling over 3 flows sharing 2 bottleneck links (constraints at sender and receiver side in data center fabrics), as shown in Figure 1(a). Flow f_1, f_2 share the same source and Flow f_1, f_3 share a common destination, but f_2 does not interfere with f_3. By the bandwidth constraint of the access link, during each slot a server can only send or receive 1 packet. At the beginning of slot 1 (i.e. time 0), both f_1 consisting of 5 packets and f_2 with 1 packet arrives. Then at the beginning of slot 2 (i.e. time 1), another 1-packet f_3 is ready. Given a scheduling algorithm, then the queue evolution process is available.

Figure 1(b) depicts the corresponding flow operations under SRPT scheduling. During the first slot, f_2 leaves, since it is shorter than f_1. After f_3 comes, slot 2 will then be occupied by the one-packet f_3. In this way, after 6 slots goes by, f_1 only transfers 4 packets and 1 packet remains.

We can see that in SRPT, why 1 packet left lies in the two 1-packet flows not overlapping in time domain. They preempt 2 slots from f_1 one after another and hence f_1 do not complete its transfer. Note that during the whole 6 slots, totally 6 packets arrive at one bottleneck and 6 packets arrive at the other. In other words, the traffic load does not exceed link capacity, and it is the SRPT scheduling algorithm that is responsible for the left packet. In real data centers, the number of competing flows is far more than 3, therefore the non-overlapped packets of small flows could preempt more slots. As a result, the remaining part of a large flow is likely to be too long to compete with those small ones. In this context, the remaining packets could not leave the queue until the next larger competing flow comes. However, the newcoming large flow will suffer severer preemption because the grabbers not only include small flows, but also the remaining part of the left one. As time goes by, the total backlog at source will become larger and larger, leading to instability. The consequence is

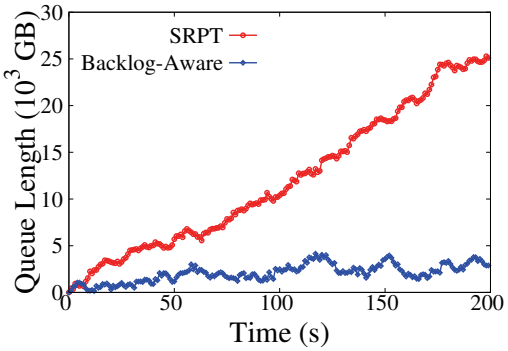


Fig. 2: Queue length at a port

that more and more flows become uncompleted even when the link capacity is enough to support the traffic load, which indicates the inefficiency of resource utilization. Moreover, the application performance will also be affected.

Actually, if the scheduling algorithm is backlog-aware, namely, considers not only the flow size but also the queue length, this problem can be totally avoided. A backlog-aware algorithm will give a large flow some slots if the backlog of the same source and destination has grown to a certain degree, which prevents more and more packets of large flows from being left. Figure 1(c) shows the result of a backlog-aware scheduling strategy. Although f_1 is larger than other flows, considering its backlog is also large, and to keep the balance of queue, the scheduler assigns the first slot to f_1. After that, the backlog only contains 4 packets and the priority of f_1 decreases, thus the subsequent slot are assigned to the two 1-packet flows. Since there is no mutual interference between f_2 and f_3, the 2 short flows complete transmission in 1 slot and the remaining 4 slots are all left to f_1. In this way, all flows complete transfer within 6 slots at the cost of 1 slot of delay increase on f_3. If this increase is acceptable, the backlog-aware strategy improves the throughput by $\frac{1}{6}$ pkt/slot, relative to SRPT. This is only a simple illustration, in practice the relative weight between flow size and queue length should be carefully assigned to achieve a suitable tradeoff. However, the simple idea of backlog-aware scheduling motivates this work.

The recent study of data center traffic explores the traffic pattern supporting data mining tasks in data centers [16]. In short, jobs relying on large chunk exchanges will travel locally, while the small queries and their responses travel across the whole cluster. The two patterns correspond to the above scenario where small and large flows coexist and the large ones have concentrated spatial distribution, hence the instability phenomenon could indeed happen in realistic applications.

Figure 2 shows the queue length variation of a simulation running SRPT and a backlog-aware strategy in the data center fabric (fat-tree topology) using the traffic pattern mentioned above. The results confirm our prediction. In the simulation, each ingress/egress link holds about 9.20 Gbps of traffic load, of which the largest one does not exceed 9.5 Gbps. Compared

with SRPT, the backlog-aware strategy just priorities flows in the backlog exceeding a certain threshold and other flows are still scheduled according to SRPT. Both algorithms are centralized, and the queue length depicted is from one of the servers. It is not hard to see that although the traffic loads at all ingress/egress links are within 10 Gbps of link capacity, the queue length under SRPT still keeps increasing, while in the backlog-aware strategy stabilizes at a certain value.

Our motivation in this work is mainly to propose a backlog-aware flow scheduling algorithm that maintains relatively low FCT while stabilizing queue length. We start by abstracting out the data center fabric into an input queued switch. Then we use the Lyapunov optimization theory [17] to design an algorithm that not only achieves good delay performance but also keeps queue stable as long as the traffic load is within network capacity. By doing this, we introduce the stability analysis into flow scheduling issues in data center networks.

III. Network Model

In this section, we develop a formalized model of queue length evolution with flow scheduling in the data center fabric, which provides a framework to serve as the foundation for further improvements. Note that the abstraction is used to simplify our analysis, but is not enforced in our simulations (Section V).

A. Data Center Fabric Abstraction and SRPT operation

Just like in quite a few data center transport designs [3, 12–15], the entire data center fabric in our analysis is abstracted out as one non-blocking input queued switch interconnecting all servers, as shown in Figure 1. It is known that in a data center fabric, any two servers could intercommunicate, and the big switch abstraction firstly guarantees this fact. Furthermore, advances in full bisection bandwidth topologies [19, 20] and techniques for implementing edge constraints into the network [13, 21] push the bottleneck to source and receiver side, which makes the abstraction reasonable. In the big switch, each port represents one server. Each ingress port has one or more flows destined to various egress ports, so it is convenient to organize the flows in *Virtual Output Queues* at ingress ports, as illustrated in Figure 1. In a data center network connecting N servers, there should be N^2 virtual queues – N at each ingress port – with each $queue\ i, j$ containing flows arriving at ingress port i and destined for egress port j, where $i, j \in \{1, \cdots, N\}$.

Like in many previous transport designs [2–5, 12], we assume that the prior knowledge of flow sizes is available, thus could be used for scheduling. In the SRPT scheduling algorithm, the scheduler identifies all active flows. The globally shortest flow is first included, and if it lies in $queue\ i, j$, then all other flows with ingress port i or egress port j are blocked and thus removed from the candidate set. Repeat the action for the rest of flows until no flow could be added, then all the selected flows compose a current scheduling decision.

B. System Operations

Assume the system operate in slotted time, the evolution process could be expressed using a discrete time stochastic process. As long as the granularity of time slots is appropriately selected, the accuracy of analysis would not be seriously violated by discretization. Because what we care about are long-time-scale properties, it is not necessary to elaborately describe the system behavior at every time point. Supposing that packets are of the same length, and let the transmission delay of a packet at its ingress port be a time slot. Obviously, the granularity of a time slot is at least several times smaller than FCT, guaranteeing the accuracy of discretization. Then due to crossbar constraints, during one time slot at most one packet can be transferred from each ingress port and at most one packet can be received by each egress port. We assume that flows randomly arrive at the ingress ports only at the end of each slot.

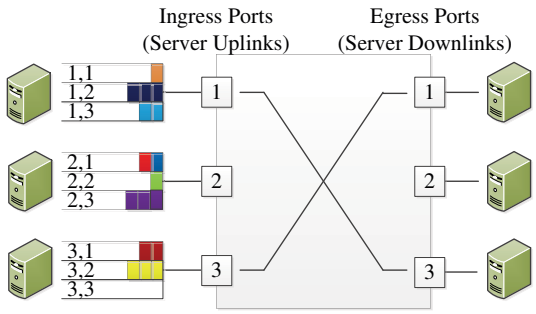


Fig. 3: Abstraction of data center fabric

In an input-queued switch model with N ingress/egress ports, q_{ij} $(i, j \in \{1, \cdots, N\})$ denotes the jth virtual queue at the ith ingress port. As stated above, it contains the flows arriving at ingress port i and destined for egress port j. When designing a size-based flow scheduling algorithm, to ensure the known flow sizes before scheduling we assume packets of one flow (however long) arrive all at once, or else the scheduler cannot know about all flow sizes on each time slot. We denote a flow by $A_{ij}(t)$, meaning the number of packets arriving at q_{ij} at the end of slot t. If no flow arrives, then $A_{ij}(t) = 0$. Since each time slot is really small relative to FCT, we could ignore the situation where more than one flows arrive at the same q_{ij} at the same slot. Due to the randomness of flow sizes, every $A_{ij}(t)$ should be a random positive integer that obeys a certain probability distribution. Suppose that $A_{ij}(t)$ for different i, j are mutually independent, and for each (i, j), $A_{ij}(t)$ are independent identically distributed for different t, with mean λ_{ij}. Let the input rate matrix $\Lambda = (\lambda_{ij})$. For the subsequent proof, we also assume an upper bound B for its secondary moment, that is, $\mathbb{E}[A_{ij}^2(t)] \leq B$ for all $i, j \in \{1, \cdots, N\}$ and all $t \in Z^+$. The assumption is reasonable because according to the measurement in [1], all flow lengths are within an upper bound of 50MB.

Each scheduling decision is a matching between ingress and egress ports, thus could be expressed with a permutation $\boldsymbol{\sigma} = (\sigma_1, \cdots, \sigma_N)$, where $\sigma_1, \cdots, \sigma_N$ are distinct elements of $\{1, \cdots, N\}$. Let Π denote the set of all $N!$ permutations. In

this setting, the scheduling decisions $R_{ij}(t) = I_{\{\sigma_i=j\}}$, which can only take values 0 or 1. $R_{ij}(t) = 1$ means ingress port i is connected to egress port j, that is, a flow in q_{ij} is to transmit a packet, if there is any such flow. Otherwise, $R_{ij}(t) = 0$. A system state $\boldsymbol{X}$ has the form $\boldsymbol{X} = (X_{ij} : i, j \in \{1, \cdots, N\})$, where X_{ij} refers to the length of q_{ij}. Under the assumption of unlimited queue lengths, the state space S is infinite.

The evolution of a queue over the time slot $[t, t+1)$ can be described as follows

$$X_{ij}(t+1) = X_{ij}(t) + A_{ij}(t) - R_{ij}(t) + L_{ij}(t) \tag{1}$$

where $L_{ij}(t) = (-(X_{ij}(t) - R_{ij}(t)))_+$, serves as a rectification that takes value 1 if and only if the scheduling imposes a potential departure at q_{ij} during slot t when $X_{ij}(t) = 0$. In the long run, the mean number of packets arriving at ingress port i during one slot is given by the sum $\sum_{j\in\{1,\cdots,N\}} \lambda_{ij}$, and similarly the $\sum_{i\in\{1,\cdots,N\}} \lambda_{ij}$ is the mean number of arriving packets destined for egress port j. The crossbar constraints impose the following necessary conditions for stability.

$$\begin{cases} \sum\limits_{j\in\{1,\cdots,N\}} \lambda_{ij} \le 1 \quad for\ all\ i \\ \sum\limits_{i\in\{1,\cdots,N\}} \lambda_{ij} \le 1 \quad for\ all\ j \end{cases} \tag{2}$$

The implication is oblivious that the queue backlog at a port will absolutely keep increasing in the long term if its mean arriving rate exceeds the link capacity, which indicates the instability.

Since the scheduling decisions are only based on current flow lengths, the next system state is irrelevant to historical information. Therefore, the evolution can be exactly described by an irreducible *Discrete Time Markov Chain* (DTMC) and the theorems for DTMC recurrence could be directly used for stability analysis [22].

Let P denote the DTMC describing the queue length evolution. Following the formal definition in [22], the stability means the recurrence of P, and the formal definition is that for every state α in S, $E_\alpha[\eta_\alpha] = \infty$, where $E_\alpha[\eta_\alpha]$ is the expectation of the number of visits to state α when starting from α itself. That is to say, once departing from a recurrent state, the DTMC will revisit it with probability 1. A DTMC is called recurrent if its every possible state is recurrent.

IV. Backlog-Aware SRPT

Based on the model given in previous section, in this section, we develop an improved algorithm called Backlog-Aware SRPT (BASRPT) for flow scheduling in data center fabrics. BASRPT is a backlog-aware version of SRPT whose design is based on the Lyapunov optimization theory. Compared with naive SRPT scheduling algorithm, BASRPT takes into account all queue lengths when making each scheduling decision. In this way, BASRPT guarantees queue stability while still maintaining the low delays properties as SRPT as long as the condition (2) is satisfied.

A. BASRPT Overview

In the context of input-queued switch, BASRPT tries to solve the following optimization problem when making each scheduling decision

$$\begin{array}{ll} Minimize: & V\bar{y}(t) - \sum_{ij} X_{ij}(t)R_{ij}(t) \\ Subject to: & R_{ij}(t)R_{ik}(t) = 0 \quad i,j,k \in \{1,\cdots,N\}, k \ne j \\ & R_{ij}(t)R_{kj}(t) = 0 \quad i,j,k \in \{1,\cdots,N\}, k \ne i \\ & No\ flows\ could\ be\ added\ any\ more \end{array}$$

$V \ge 0$ is an importance weight on how much we emphasize average FCT minimization, similar to the tradeoff parameters in [23]. $\bar{y}(t)$ is the mean length of all flows selected by the current scheme and $\sum_{ij} X_{ij}(t)R_{ij}(t)$ is the sum of selected queue lengths. $R_{ij}(t)$ is an indicator whether any flows in q_{ij} is scheduled on slot t, then the first two conditions explain the crossbar constraint that each ingress/egress port can appear at most once in a scheduling decision. The last constraint means the flow selection should be in a maximal manner. The scheduling decision is updated when a flow comes or a transfer completes.

To achieve the optimization objective, BASRPT each time iterates through all possible scheduling schemes and chooses the one with the least $V\bar{y}(t) - \sum_{ij} X_{ij}(t)R_{ij}(t)$. Intuitively, the weighted sum expresses the expect of choosing small flows in long queues, which corresponds to the goal of reducing FCT while stabilizing queue. Different from SRPT, BASRPT comprehensively considers flow size and queue length at execution. If we set V to be arbitrarily large, then the expectation $\mathbb{E}[\bar{y}(t)]$ is to be pushed arbitrarily close to the optimal value, and BASRPT would degenerate to SRPT.

It is apparent that to stabilize a queue in the long run, the average input rate must be no more than the average output rate, or else the backlog will keep accumulating. Because of the existing possibility for scheduling empty queues, the actual output rate should be less than or equal to the average scheduling rate $\bar{R}_{ij}$, then a stable scheduling algorithm must firstly make sure that $\lambda_{ij} \le \bar{R}_{ij}$ for all $i, j \in \{1, \cdots, N\}$.

Consider first an algorithm satisfying the above rate requirement. For the input rate matrix Λ, a line sum is either a row sum $\sum_j \lambda_{ij}$ or a column sum $\sum_i \lambda_{ij}$. Under the necessary condition for stability, any $\sum_j \lambda_{ij}$ should be less than or equal to 1 pkt/slot, and the same restriction applies to any $\sum_i \lambda_{ij}$. Thus by appropriately increasing some of the entries of Λ we could get a doubly stochastic matrix M, and $\lambda_{ij} \le M_{ij}$ for all i, j. A doubly stochastic matrix is a square matrix whose entries are nonnegative and all its line sums are equal to 1. In terms of Birkhoff's theorem [24], M can be expressed as a convex combination of permutation matrices, that is, $M = \sum_{\boldsymbol{\sigma}\in\Pi} M(\boldsymbol{\sigma})u(\boldsymbol{\sigma})$. $M(\boldsymbol{\sigma})$ represents the $N \times N$ permutation matrix whose entries totally depend on the permutation $\boldsymbol{\sigma}$. If $\sigma_i = j$, then $M_{ij} = 1$, or else $M_{ij} = 0$. $u(\boldsymbol{\sigma})$ is the appearance probability of the permutation $\boldsymbol{\sigma}$ and $\sum_{\sigma\in\Pi} u(\boldsymbol{\sigma}) = 1$.

Summarizing the above statements, there is a probability distribution $\boldsymbol{u}$ on Π so that for all $i, j \in \{1, \cdots, N\}$, $\lambda_{ij} \le \sum_{\sigma\in\Pi} M(\sigma)u(\sigma)$. This fact means, as long as the scheduling

decision obeys the probability distribution $\boldsymbol{u}$, the requirement $\lambda_{ij} \leq \bar{R}_{ij}(t)$ would be satisfied, and define ϵ as follows:

$$\epsilon = \max\{\epsilon' : \lambda_{ij} + \epsilon' \leq \bar{R}_{ij}\} \quad for\ all\ i, j$$

There could be various methods to realise the above distribution. For example, the scheduler can choose the permutation from Π according to distribution $\boldsymbol{u}$ when making each scheduling decision, or give each σ a time period and let the proportion the period accounts for to be $u(\sigma)$. From all scheduling algorithms satisfying this probability distribution, there should be an optimal one in the sense of minimizing the time average of $\bar{y}(t)$, here we define it as α^*.

However, only meeting the above condition is not enough to ensure stability, hence BASRPT further draws upon the Lyapunov optimization method to guarantee the positive recurrence as well as the bounded delay. The detailed proof is in subsection IV-B.

B. Analysis of BASRPT

In fact, BASRPT is a normal example of *drift-plus-penalty* method in Lyapunov optimization. To prove either stability or delay performance of BASRPT, we need to firstly define the Lyapunov function, which directly determines algorithm performance.

Unfortunately, the selection of Lyapunov functions has no fixed methods so far, and existing works often do it empirically. Since the subsequent analysis needs the Lyapunov function to be a non-negative function of system states, here we define a quadratic Lyapunov function $L(\boldsymbol{X}(t))$ as follows:

$$L(\boldsymbol{X}(t)) \triangleq \frac{1}{2}\sum_{i,j} X_{ij}^2(t) \tag{3}$$

Obviously, the definition of quadratic sum of all queue lengths ensures non-negativity, and represents a scalar measure of queue congestion in networks. In addition, $L(\boldsymbol{X}(t)) = 0$ occurs if and only if all queues are empty at slot t. So it is intuitively clear that an algorithm concentrating on pushing $L(\boldsymbol{X}(t))$ towards zero will help to stabilize queue and avoid congestion.

After defining the Lyapunov function, we could then use it to get another important variable, that is the Lyapunov drift $\Delta(\boldsymbol{X}(t))$. Literally, the Lyapunov drift is the difference between the Lyapunov function values of two subsequent slots, and the strict definition is as follows:

$$\Delta(\boldsymbol{X}(t)) \triangleq \mathbb{E}[L(\boldsymbol{X}(t+1))|\boldsymbol{X}(t)] - L(\boldsymbol{X}(t)) \tag{4}$$

where $\mathbb{E}[L(\boldsymbol{X}(t+1))|\boldsymbol{X}(t)]$ denotes the conditional expectation of the Lyapunov function at slot $(t+1)$ under the present state $\boldsymbol{X}(t)$. The definition is from the perspective of slot t. Because of the randomness of flow arrivals, we cannot know the system state on slot $(t+1)$ deterministically, which explains the conditional expectation taken in the above definition.

Another variable remaining to be defined in the optimization problem in Section IV-A is $\bar{y}(t)$, which is the metric whose time average we want to optimize besides stability. In the *drift plus penalty* method [17], $\bar{y}(t)$ is called "penalty", and "optimize" must be "minimize", hence those metrics waiting to be maximized must experience the transformation, such as taking reciprocals or opposite numbers. In our context, there is no doubt that what we want to minimize is the average FCT. However, the FCT cannot be measured before the flow transfer completes, that is to say, we cannot get this metric when making scheduling decisions. In SRPT, the goal of minimizing FCT is simply realized by choosing as short flows as possible. Similarly, in our design, the metric of FCT could also be transformed into the average of selected flow lengths. The average value rather than the sum is to avoid the preference for scheduling with less flows which lowers the link utilization. In this case, the "penalty" $\bar{y}(t)$ is defined as the average length of selected flows, that is

$$\bar{y}(t) \triangleq \frac{\sum_{f\in s(t)} y_f(t)}{|s(t)|}$$

where $y_f(t)$ denotes the remaining size of flow f and $s(t)$ is the set of selected flows on slot t.

Theorem 1. *BASRPT is stable, and the difference between the time average expected $\bar{y}$ of BASRPT and that of the delay-optimal algorithm α^* is bounded by $\frac{N(1+NB)}{2V}$.*

Proof. Substituting (1) and (3) into the expression of $\Delta(\boldsymbol{X}(t))$, we could get

$$\begin{aligned}\Delta(\boldsymbol{X}(t)) =& \frac{1}{2}\sum_{i,j}\mathbb{E}[(X_{ij}(t) + A_{ij}(t) - R_{ij}(t) + L_{ij}(t))^2|\boldsymbol{X}(t)] \\ &-\frac{1}{2}\sum_{i,j} X_{ij}^2(t) \leq \sum_{i,j} X_{ij}(t)\mathbb{E}[A_{ij}(t) - R_{ij}(t)|\boldsymbol{X}(t)] \\ &+\frac{1}{2}\sum_{ij}\mathbb{E}[A_{ij}^2(t) + R_{ij}^2(t)|\boldsymbol{X}(t)]\end{aligned}$$

where the final inequality is in terms of the fact that for any $X_{ij}(t) \geq 0, R_{ij}(t) \geq 0$ and $A_{ij}(t) \geq 0$,

$$\begin{aligned}&(X_{ij}(t) + A_{ij}(t) - R_{ij}(t) + L_{ij}(t))^2 \leq (X_{ij}(t) + A_{ij}(t) - R_{ij}(t))^2 \\ &= X_{ij}^2(t) + (A_{ij}(t) - R_{ij}(t))^2 + 2X_{ij}(t)(A_{ij}(t) - R_{ij}(t)) \\ &\leq X_{ij}^2(t) + A_{ij}^2(t) + R_{ij}^2(t) + 2X_{ij}(t)(A_{ij}(t) - R_{ij}(t))\end{aligned}$$

The first inequality is because the variable $L_{ij}(t)$ can only push $X_{ij}(t) + A_{ij}(t) - R_{ij}(t)$ closer to zero, and the final inequality is due to the remove of the non-positive item $-2A_{ij}(t)R_{ij}(t)$.

As the arrivals $A_{ij}(t)$ are i.i.d. over time slots, they should be independent of the current queue backlogs $\boldsymbol{X}(t)$, that is $\mathbb{E}[A_{ij}(t)|\boldsymbol{X}(t)] = \mathbb{E}[A_{ij}(t)] = \lambda_{ij}$. Hence we have

$$\begin{aligned}\Delta(\boldsymbol{X}(t)) \leq& \frac{1}{2}\sum_{ij}\mathbb{E}[A_{ij}^2(t) + R_{ij}^2(t)|\boldsymbol{X}(t)] \\ &+\sum_{i,j} X_{ij}(t)\lambda_{ij} - \sum_{i,j} X_{ij}(t)\mathbb{E}[R_{ij}(t)|\boldsymbol{X}(t)]\end{aligned} \tag{5}$$

Since $\mathbb{E}(A_{ij}^2(t)) \leq B$ for all $i, j \in 1, \cdots, N$ and all $t \in Z^+$ (seen in subsection III B), we can also give an upper bound of the first term on the right-hand-side of the above drift inequality. As stated in the above section, due to the crossbar constraint, there could be at most N $R_{ij}(t)$ equal to 1 in a scheduling decision, and others must be 0, therefore $\frac{1}{2}\sum_{i,j}\mathbb{E}[R_{ij}^2(t)|\boldsymbol{X}(t)] \leq \frac{N}{2}$, thus for all t, all possible $\boldsymbol{X}(t)$

and all possible scheduling decisions that can be taken, we have:

$$\frac{1}{2}\sum_{i,j}\mathbb{E}[A_{ij}^2(t)+R_{ij}^2(t)|\boldsymbol{X}(t)] \le \frac{N}{2}(1+NB)$$

Let $B'=\frac{N}{2}(1+NB)$ and substitute it into (5), we could get

$$\Delta(\boldsymbol{X}(t)) \le B' + \sum_{i,j} X_{ij}(t)\lambda_{ij} - \sum_{i,j} X_{ij}\mathbb{E}[R_{ij}(t)|\boldsymbol{X}(t)] \quad (6)$$

Adding the item $V\mathbb{E}[\bar{y}(t)|\boldsymbol{X}(t)]$ on both sides of (6) yields

$$\begin{aligned}\Delta(\boldsymbol{X}(t)) + V\mathbb{E}[\bar{y}(t)|\boldsymbol{X}(t)] \le & B' - \sum_{ij}\mathbb{E}[X_{ij}(t)R_{ij}(t)|\boldsymbol{X}(t)] \\ & + V\mathbb{E}[\bar{y}(t)|\boldsymbol{X}(t)] + \sum_{ij} X_{ij}(t)\lambda_{ij}\end{aligned}$$

B' is a constant, and the second item on the right-hand side $\sum_{i,j} X_{ij}(t)\lambda_{ij}$ is also a fixed value under given $\boldsymbol{X}(t)$. Thus by the concept of *opportunistically maximizing an expectation* [17], the action of minimizing $V\bar{y}(t) - \sum_{ij} X_{ij}(t)R_{ij}(t)$ actually minimizes the whole right-hand side, a bound on the drift-plus-penalty.

In this way, if we replace BASRPT with any other scheduling algorithm, including α^*, the bound will certainly increase, or at least stay unchanged. That is,

$$\begin{aligned}\Delta(\boldsymbol{X}(t)) + V\mathbb{E}[\bar{y}(t)|\boldsymbol{X}(t)] \le & B' - \sum_{ij} X_{ij}(t)\mathbb{E}[R_{ij}^*(t)|\boldsymbol{X}(t)] \\ & + V\mathbb{E}[\bar{y}^*(t)|\boldsymbol{X}(t)] + \sum_{ij} X_{ij}(t)\lambda_{ij}\end{aligned}$$

where $\Delta(\boldsymbol{X}(t)) + V\mathbb{E}[\bar{y}(t)|\boldsymbol{X}(t)]$ is the drift-plus-penalty of BASRPT and $\bar{y}^*(t)$, $R_{ij}^*(t)$ are both the result of α^* on slot t. We suppose the expected penalty $\mathbb{E}[y(t)]$ is lower bounded by a finite value y_{min} and that $\mathbb{E}[L(\boldsymbol{X}(0))<\infty]$ which are easy to satisfy in reality. Note that α^* does not take into account the queue backlogs when scheduling flows, we have $\mathbb{E}[\bar{y}^*(t)|\boldsymbol{X}(t)] = \mathbb{E}[\bar{y}^*(t)]$. With the definition of ϵ, from above inequality we can obtain

$$\Delta(\boldsymbol{X}(t)) + V\mathbb{E}[\bar{y}(t)|\boldsymbol{X}(t)] \le B' + V\mathbb{E}[\bar{y}^*(t)] - \epsilon\sum_{ij} X_{ij}(t) \quad (7)$$

According to the *Lyapunov Optimization Theorem* [17], since there are constants $B'\ge 0, V\ge 0, \epsilon\ge 0$ and $\mathbb{E}[\bar{y}^*(t)]$ making the inequality (7) true for all slots $\{0,1,2,\cdots\}$ and all possible $\boldsymbol{X}(t)$, we readily have: If $\epsilon>0$ then the evolution process is positive recurrent, and if $V>0$, the time average of both expected penalty $\mathbb{E}[\bar{y}(t)]$ and queue lengths $\sum_{ij}\mathbb{E}[X_{ij}]$ satisfy:

$$\lim_{t\to\infty}\frac{1}{t}\sum_{\tau=0}^{t-1}\mathbb{E}[\bar{y}(\tau)] \le \mathbb{E}[\bar{y}^*(t)] + \frac{B'}{V}$$

$$\lim_{t\to\infty}\frac{1}{t}\sum_{\tau=0}^{t-1}\sum_{ij}\mathbb{E}[|X_{ij}(\tau)|] \le \frac{B'+V(\mathbb{E}[\bar{y}^*(t)]-y_{min})}{\epsilon}$$

□

Discussion: The above conclusion indicates, the queue lengths in the system is stable and that the average expected queue backlog are upper bounded as $O(V)$. Furthermore, the difference between the delay performance of BASRPT and the delay-optimal algorithm is also bounded by $\frac{B'}{V}$. In this case, we choose the V value by the requirements to performance and steady queue length. However, the necessary condition for stability also includes the case of $\lambda_{ij}=1$, in other words, $\epsilon=0$, which the theorem does not cover. But by careful investigation, we can find any scheduling algorithm cannot ensure the recurrence if the special case really occurs. In this special case, there must exist an ingress/egress port whose input rate is 1 pkt/slot, even if a scheduler could allow one packet to leave on every slot, there could be wasted slots when the whole ingress/egress port is empty. If the traffic contains serious burstiness, the total queue length on this port is likely to stay around a large value, directly violating the recurrence. However, from another point of view, the real data center networks usually do not work at full bandwidth, thus the special case rarely occurs.

C. Practical and Fast BASRPT Implementation

While BASRPT theoretically guarantees both the delay performance and stability, it has a tough problem with implementation. Because BASRPT does not assign priorities to flows, no distributed designs could be used for implementation. Thus when making each scheduling decision, the BASRPT scheduler must traverse all possible maximal sets of the current non-empty queues. The number of such sets could add up to $N!$ if all queues are not empty. In today's data centers, N could be really large. The large N implies the substantial possible maximal sets that the scheduler needs to traverse. And in each possible scheduling BASRPT needs to compute the weighted sum $V\bar{y}(t)-\sum_{ij}X_{ij}(t)R_{ij}(t)$. To achieve this, the scheduler counts the number of selected flows and calculates the sum of their sizes as well as the total length of the queues where they are located, which also involves unneglectable computational overhead. Many applications involve numerous small flows (e.g. queries), hence the events of flow arrivals and completions could occur so frequently that incurs also frequent scheduling updates. In this context, the huge computational overhead makes BASRPT impractical.

To reduce the implementation complexity, we develop fast BASRPT algorithm, a heuristic approach to approximate BASRPT and ensure the acceptable computation overhead. Similar to the SRPT implementation in the data center fabric, fast BASRPT also selects flows one by one, and the only difference lies in the selection rules. SRPT scheduler chooses the shortest flow from current available ones each time, while the fast BASRPT scheduler chooses the flow with the smallest weighted sum $\frac{V}{N}\times y(t)-X_{ij}(t)$. Here, $y(t)$ is the flow size and $X_{ij}(t)$ represents length of the queue where the flow resides. Note that the weight coefficient is $\frac{V}{N}$, the weight in BASRPT divided by the number of servers N. On account of the crossbar constraint, a scheduling could contain at most N flows and here we use N to approximate the number of selected flows $n(t)$ in each scheduling. Because we do not know how many flows are included in a scheduling decision before we really construct it, we need to make an estimation during the flow selection. In this way, adding up the weighted sums of all flows in a scheduling yields $\frac{V}{N}\sum_{f\in\mathcal{S}}y(t)-\sum_{ij}X_{ij}(t)R_{ij}(t)$,

where $\mathcal{S}$ is the set of flows involved in the current scheduling. Since N is used to approximate $n(t)$, the obtained sum could be considered as $V\bar{y}(t) - \sum_{ij} X_{ij}(t)R_{ij}(t)$, which is identical to BASRPT's optimization goal. The pseudo code is provided in Algorithm 1. Each time making a decision, at most N^2 flows are sorted, hence the running time should be $O(N^2 \times \log(N^2)) = O(N^3)$.

Algorithm 1 Fast BASRPT algorithm

Input: $\mathcal{F}$ = Set of active flows with their located queue, ingress port, egress port, and remaining size
Output: $\mathcal{D}$ = Set of selected flows

$D \leftarrow \emptyset$
$ingressBusy[1, \cdots, N] \leftarrow FALSE$
$egressBusy[1, \cdots, N] \leftarrow FALSE$
for each $f \in \mathcal{F}$, in non-decreasing order of $\frac{V}{N} \times remaining\ size - located\ queue\ length$ **do**
  if $ingressBusy[f.ingress] == FALSE$ **and** $egressBusy[f.egress] == FASLE$ **then**
    D.ADD(f)
    $ingressBusy[f.ingress] \leftarrow TRUE$
    $egressBusy[f.egress] \leftarrow TRUE$
  end if
end for
return $\mathcal{D}$

V. Evaluation

In this section, we conduct flow-level simulations to evaluate the performance of our algorithm. The traffic distribution used in the simulation is constructed depending on experimental observations in [1]. We first evaluate both FCT and queue stability, taking SRPT as reference. Then we discuss the influence of different values of V on fast BASRPT performance. The simulation parameters mainly refer to those in [3].

A. Simulation Methodology

Platform and Topology: We develop our own flow-level simulator in Java. For the topology, we use the multi-rooted hierarchical tree topology shown in Figure 4, which is a common case for data center topology [19, 25]. The fabric contains 3 layers and interconnects 144 hosts through 12 top of rack (ToR) switches and 3 core switches, and all of them compose a full connection. The 144 hosts are evenly distributed over 12 racks, therefore each rack holds 12 hosts. Each host is connected to a ToR switch by a 10Gbps link, while each ToR switch simultaneously connects to all 3 core switches with 40Gbps links. Such bandwidth configuration guarantees the bottleneck not to be in network.

Workloads: The workload is generated following the statistical results given in recent data center traffic measurements. Literature [1] provides the distribution of both flow inter-arrival times and flow sizes. The workloads have obvious heavy-tailed characteristics, and over 95% of all bytes are from the 30% of flows with the size of 1-20 MB [3]. The measurement conducted on the spatial distribution of traffic

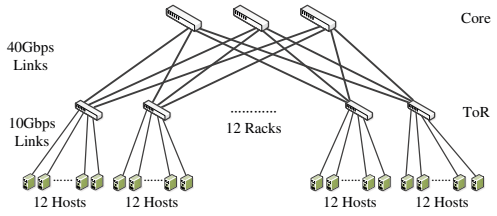


Fig. 4: Simulation topology

discovers two flow patterns in the data centers [16]. In brief, large transfers (e.g. backups, backend operations) usually travel within a rack, while small flows (queries and responses) across the whole fabric.

In simulations, each server concurrently holds small and large flows. Since queries and responses are of fixed size in practice, we set them to be 20KB. These small flows follow Poisson arrival process, and for each source the destinations are uniformly chosen from all 144 hosts at random. As for background flows, the inter-arrival times and flow sizes both follow the distributions in [1]. Considering the spatial locality, the destination of each background flow uniformly falls in the same rack with its source. Finally, for whatever kind of flows, the arrival rates vary to achieve a desired level of load in fabric.

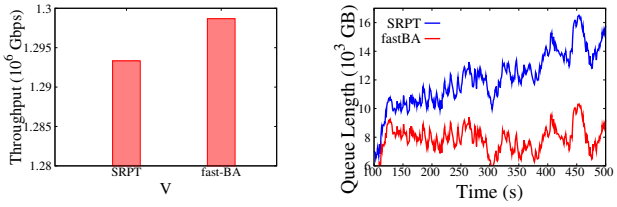
In order to study the stability property, we focus on the traffic nearly saturating network but carefully control the volume between each server pair so that the workload on each port does not exceed link capacity. In this way, we can observe the performance of fast BASRPT under the stressed network conditions. For our 10Gbps of port link bandwidth, we generate around 9.5Gbps of loads on each ingress/egress port. However, to evaluate delay performance we also involve simulations under loads from 1Gbps to 8Gbps.

Performance Metrics: To study the delay performance and stability properties, we consider 3 main metrics:

- **FCT**. We separately compute that value for small and large flows. Furthermore, for queries and responses, we calculate the 99th percentile FCT, since it is often those tardy flows that affect the application performance most.
- **Throughput**. Throughput here is calculated globally in bytes, counting the total data volume leaving the fabric during the whole simulation period. At the end of a simulation, those packets not leaving are left in the fabric. Throughput directly reflects the link utilization, the metric that BASRPT optimizes besides FCT.
- **Queue length variation trend**. Considering that the stability is a metric on infinite time scales, we observe the evolution trends of queue lengths over a period of 500s, which is long enough to judge whether a queue is stable or not. Because the long period filters out the impact of short-term arrivals, if the queue length keeps growing in macroscale during the total 500s, we think of it as unstable.

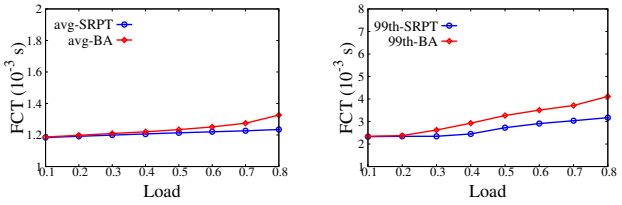
TABLE I: Average FCT and 99th percentile FCT (ms)

	query: Avg	query: 99th	back: Avg	back: 99th
fast BA	2.49	14.18	25.04	406.49
SRPT	1.34	4.03	26.76	403.58



(a) Throughput (b) Typical queue length

Fig. 5: Throughput and queue length



(a) Average FCT (b) 99th FCT

Fig. 6: Varying Loads

B. Performance Measures

Fast BASRPT achieves good delay performance, obtains a higher throughput, and stabilizes the queue at an acceptable level in our simulations. Table 1 shows the average and 99th percentile FCT for both queries (responses) and background flows, taking as reference SRPT. We observe that, for queries the average FCT in fast BASRPT is less than $2\times$ SRPT and the 99th percentile is less than $4\times$ SRPT, while for background flows both the average and the 99th percentile are basically consistent with SRPT. We know that the pFabric developed in [3] totally adopts the SRPT flow scheduling and achieves near-optimal flow completion times for all flow sizes. Thus the $2\times$ average FCT as well as $4\times$ 99th percentile do not increase too much. For throughput and queue length stabilization, Figure 5(a) depicts the global throughput of the two schemes in the first 500 seconds, and Figure 5(b) exhibits the evolution of a typical queue. In SRPT, the queue length keeps growing all the time, while in fast BASRPT the queue stabilizes at 8×10^3 GB. Besides, by statistics, the global throughput improves by 5352 Gbps compared with SRPT. Here we just choose $V = 2500$ for demonstration, and if V takes a larger value, according to Theorem 1 the FCT will become even smaller.

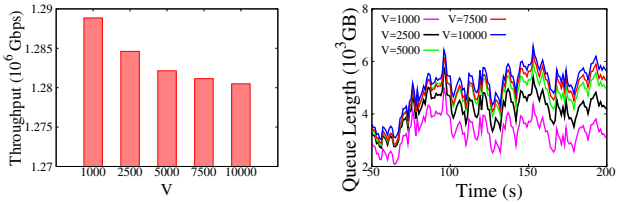
To evaluate the fast BASRPT performance at various traffic loads, Figure 6 compares the average FCT, 99 percentile FCT for queries and overall throughput of the two schemes as we vary the traffic load from 10% to 80%. We observe that when the load is low, the instability nature of SRPT does not exhibit and fast BASRTP shows almost identical FCT as well as throughput performance as SRPT. As the load grows, the average FCT of both SRPT and fast BASRPT increases modestly, but the growth degree in fast BASRPT is a little larger. Note that even at 80% of load, the average and 99th percentile FCT of fast BASRPT only increase by 7.4% and 29.7% compared with SRPT. On the other hand, the overall throughput in fast BASRPT is a little higher under all load conditions. That is to say, BASRPT provides a flexible scheduling. When the traffic load is relatively low, BASRPT performs just like SRPT and achieves near-optimal FCT performance. And when the load increases, BASRPT supplements SRPT and provides the stability guarantee.

C. Impact of Parameter V

Making V vary from 1000 to 10000, we show the throughput and queue length evolution in Figure 7 and the average as well as 99th percentile FCT in Figure 8. The large V is because the two properties in the optimization objective are the average selected flow size and the total queue length respectively. Obviously, the latter is much larger and we need to give the former a large weight. The results verify the role of parameter V, a weight on FCT minimization against stabilizing queue. As V becomes larger, both average and 99th percentile query FCT sees significant reduction, while the stable point of queue length goes up slightly. However, for background flows the average FCT and the 99th percentile present different trends. Relative to queries, background flows are larger, thus they are deprived of more slots and the average FCT goes up. But according to [1], most background flows are also small. When V gets larger, flows at the 99th percentile could also preempt more slots from those smaller background flows, in which case the 99th percentile FCT slightly decreases. Furthermore, the global throughput also sees a slight decline, that is because the scheduler lays more emphasis on delay performance and thus increases the stable queue length. However, we can see the variation of V does not make a big difference on queue lengths and throughput, but greatly improves the FCT performance, which indicates the good properties of fast BASRPT.

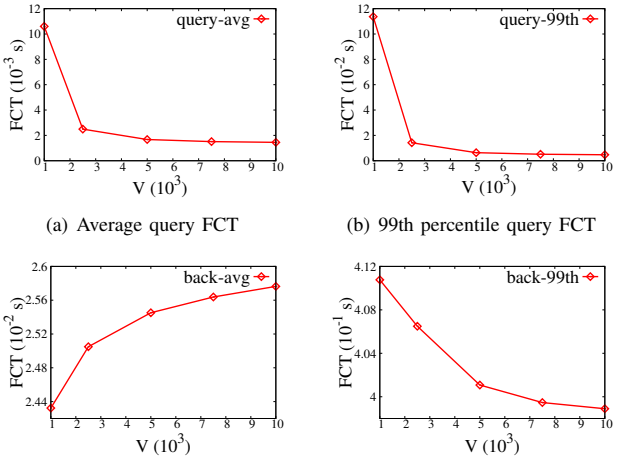
VI. Conclusion

In this paper we developed BASRPT, a backlog-aware flow scheduling algorithm in data center fabrics. Based on Lyapunov optimization theory, we prove that the queue evolution process is recurrent and the FCT could be pushed arbitrarily close to optimal value within the range of stability. We also proposed fast BASRPT, a fast and practical approximation of BASRPT for reducing computational overheads. We perform extensive flow-level simulations to evaluate our proposed approach, leveraging real data center workloads and traffic patterns. We find that fast BASRPT stabilizes the queues as long as the traffic loads are within network capacity and meanwhile achieves a relatively low FCT. Besides, the global throughput is also improved. BASRPT provides a flexible framework to better adapt to different performance requirements, since we can balance delay and link utilization by setting different V.



(a) Throughput (b) Typical queue lengths

Fig. 7: Throughput and queue lengths under different V



(a) Average query FCT (b) 99th percentile query FCT

(c) average back FCT (d) 99th percentile back FCT

Fig. 8: FCTs under different V

VII. Acknowledgments

The authors gratefully appreciate our shepherd Dr. Ganesh Ananthanarayanan for his constructive suggestions, and acknowledge the anonymous reviewers for their valuable comments. This work is supported in part by National Natural Science Foundation of China (NSFC) under Grant No. 61225011, National Basic Research Program of China (973 Program) under Grant No.2012CB315803, and National High-Tech Research and Development Plan of China (863 Plan) under Grant No. 2015AA020101.

References

[1] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan, "Data center tcp (dctcp)," *ACM SIGCOMM CCR*, 2011.

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

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

[4] A. Munir, G. Baig, S. M. Irteza, I. A. Qazi, A. X. Liu, and F. R. Dogar, "Friends, not foes: synthesizing existing transport strategies for data center networks," in *Proceedings of the 2014 ACM conference on SIGCOMM*, 2014.

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

[6] L. E. Schrage and L. W. Miller, "The queue m/g/1 with the shortest remaining processing time discipline," *Operations Research*, 1966.

[7] S. Yang and G. De Veciana, "Size-based adaptive bandwidth allocation: Optimizing the average qos for elastic flows," in *INFOCOM 2002. Proceedings. IEEE*, 2002.

[8] M. Verloop, S. Borst, and R. Núñez-Queija, "Stability of size-based scheduling disciplines in resource-sharing networks," *Performance Evaluation*, 2005.

[9] T. Bonald and L. Massoulié, "Impact of fairness on internet performance," in *ACM SIGMETRICS Performance Evaluation Review*, 2001.

[10] S. Aalto and U. Ayesta, "Srpt applied to bandwidth-sharing networks," *Annals of Operations Research*, 2009.

[11] P. X. Gao, A. Narayan, G. Kumar, R. Agarwal, S. Ratnasamy, and S. Shenker, "phost: Distributed near-optimal datacenter transport over commodity network fabric."

[12] M. Chowdhury, Y. Zhong, and I. Stoica, "Efficient coflow scheduling with varys," in *Proceedings of the 2014 ACM conference on SIGCOMM*, 2014.

[13] H. Ballani, P. Costa, T. Karagiannis, and A. Rowstron, "Towards predictable datacenter networks," in *ACM SIGCOMM CCR*, 2011.

[14] N. Kang, Z. Liu, J. Rexford, and D. Walker, "Optimizing the one big switch abstraction in software-defined networks," in *Proceedings of ACM conference on Emerging networking experiments and technologies*, 2013.

[15] L. Popa, G. Kumar, M. Chowdhury, A. Krishnamurthy, S. Ratnasamy, and I. Stoica, "Faircloud: sharing the network in cloud computing," in *Proceedings of the ACM SIGCOMM 2012 conference on Applications, technologies, architectures, and protocols for computer communication*, 2012.

[16] S. Kandula, S. Sengupta, A. Greenberg, P. Patel, and R. Chaiken, "The nature of data center traffic: measurements & analysis," in *Proceedings of ACM SIGCOMM conference on Internet measurement conference*, 2009.

[17] M. J. Neely, "Stochastic network optimization with application to communication and queueing systems," *Synthesis Lectures on Communication Networks*, 2010.

[18] A. Bar-Noy, M. M. Halldórsson, G. Kortsarz, R. Salman, and H. Shanhnai, *Sum multi-coloring of graphs*, 1999.

[19] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta, "Vl2: a scalable and flexible data center network," in *ACM SIGCOMM CCR*, 2009.

[20] R. Niranjan Mysore, A. Pamboris, N. Farrington, N. Huang, P. Miri, S. Radhakrishnan, V. Subramanya, and A. Vahdat, "Portland: a scalable fault-tolerant layer 2 data center network fabric," in *ACM SIGCOMM CCR*, 2009.

[21] A. D. Ferguson, A. Guha, C. Liang, R. Fonseca, and S. Krishnamurthi, "Participatory networking: An api for application control of sdns," in *ACM SIGCOMM CCR*, 2013.

[22] S. P. Meyn and R. L. Tweedie, *Markov chains and stochastic stability*, 2012.

[23] R. Grandl, G. Ananthanarayanan, S. Kandula, S. Rao, and A. Akella, "Multi-resource packing for cluster schedulers," in *ACM SIGCOMM CCR*, 2014.

[24] M. J. Neely, E. Modiano, and Y.-S. Cheng, "Logarithmic delay for n× n packet switches under the crossbar constraint," *IEEE/ACM Transactions on Networking (TON)*, 2007.

[25] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," *ACM SIGCOMM CCR*, 2008.