On Fairness-Efficiency Tradeoffs for Multi-Resource Packet Processing

Wei Wang, Ben Liang, Baochun Li
Department of Electrical and Computer Engineering
University of Toronto

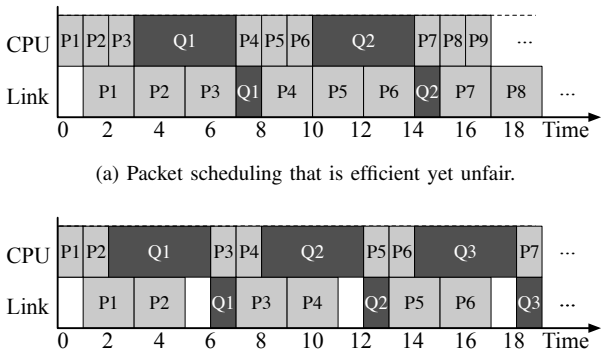
***Abstract*—Middleboxes are widely deployed in today's datacenter networks. They perform a variety of network functions, each requiring multiple hardware resources, such as CPU cycles and link bandwidth. Depending on the functions they go through, packet processing of different traffic flows may consume a vastly different amount of hardware resources. An effective algorithm is therefore highly desired to schedule packets in a way such that multiple resources are shared in a fair and efficient manner. However, we show in this paper that there exists a fairness-efficiency tradeoff when multiple resources are scheduled. Such a tradeoff has never been a problem for traditional single-resource fair queueing (*e.g.,* GPS, WFQ, SCFQ, DRR) — as long as the queueing schemes are work conserving, both fairness and efficiency can be achieved simultaneously — and hence has received little attention. Therefore, a new and important research problem arises: given a desired fairness-efficiency tradeoff, how can we design a packet scheduling algorithm to reinforce such a tradeoff? We present our thoughts and observations in this paper.**

I. Introduction

Middleboxes have found widespread adoption in today's enterprise and datacenter networks. According to [1], [2], the sheer number of middleboxes deployed is already comparable to the traditional L2/L3 infrastructures. These middleboxes perform a wide range of important network functions, such as WAN optimization, intrusion detection, and firewalls at the network or application layers.

Unlike basic forwarding, most network functions performed by middleboxes require deep packet processing based on the packet contents, and hence consume a variety of hardware resources, *e.g.,* CPU, memory bandwidth, and link bandwidth. Packet processing for these middlebox functions differs significantly in terms of the amount of hardware resources required. For example, forwarding a large amount of small packets via software routers congests the memory bandwidth [3]. The intrusion detection system, on the other hand, usually suffers bottlenecks on the CPU, as packets of external traffic flows need to be analyzed before being sent to the internal destinations. In general, depending on the network functions they go through, different traffic flows may require vastly different types and amounts of middlebox resources [4].

Having heterogeneous resource requirements among traffic flows significantly complicates resource scheduling in middleboxes. It is highly desirable to have a queueing algorithm to schedule packets in a way such that multiple middlebox resources (*e.g.,* CPU and link bandwidth) are shared among flows in a *fair* and *efficient* manner. By "fair" we mean that each flow should receive *predictable service isolation* that is independent of others' demand. By "efficient" we mean that flows should finish their services as fast as possible with maximum resource utilization.



(a) Packet scheduling that is efficient yet unfair.

(b) Packet scheduling that is fair yet inefficient.

Fig. 1. The tradeoff between fairness and efficiency when multiple resources are scheduled. Flow 1 sends P1, P2, ..., each requiring ⟨1 CPU time, 2 Transmission time⟩. Flow 2 sends Q1, Q2, ..., each requiring ⟨4 CPU time, 1 Transmission time⟩.

However, achieving both fairness and efficiency at the same time may not be possible when multiple resources are to be scheduled. In fact, there may exist a tradeoff between the scheduling fairness and the scheduling efficiency. Consider the following example. Suppose there are two traffic flows that keep sending packets. Packets of Flow 1, denoted by P1, P2, ..., need basic forwarding only. Each of them requires 1 time unit for CPU processing and 2 time units for link transmission. Packets of Flow 2 (denoted by Q1, Q2, ...), on the other hand, need encryption before transmission, each requiring 4 time units for CPU processing and 1 time unit for link transmission. Fig. 1a illustrates a schedule with the maximum resource utilizations on both CPU and link bandwidth. As we will see later in Sec. II, this schedule, though highly efficient, allocates too much resource to Flow 1, and is unfair to Flow 2. Another schedule, shown in Fig. 1b, allocates the same processing time to two flows on their most congested resources, and is fair according to the definition of Dominant Resource Fairness (DRF) [5]. However, it is inefficient as the link bandwidth is not fully utilized. As shown in Fig. 1b, it takes 17 time units for the schedule to finish the service of the first 6 packets of Flow 1 (*i.e.,* P1, ..., P6) and the first two packets of Flow 2 (*i.e.,* Q1, Q2). In comparison, the schedule shown in Fig. 1a takes only 15 time units to serve the same amount of traffic.

Such a fairness-efficiency tradeoff has received little atten-

tion in the existing fair queueing literature. When there is a single resource to schedule (*i.e.,* link bandwidth), the tradeoff does not exist — a fair queueing algorithm is also the most efficient as long as it is *work conserving* — and hence has never been a problem in the traditional fair queueing literature [6], [7], [8], [9]. Existing works on multi-resource fair queueing, (*e.g.,* [4], [10]) on the other hand, focus solely on fairness without discussing efficiency. In fact, even how the efficiency of a scheduling scheme should be measured is unclear. To our knowledge, [11] is the only work that raises attention on the fairness-efficiency tradeoff. Even there, the focus is on allocating resources where they are concurrently shared among users *in space*. However, in middleboxes, hardware resources are limited and have to be multiplexed by multiple traffic flows *in time*.

While fairness is an important concern for resource scheduling, it is by no means the only objective to pursue in all cases. Some applications may have a loose requirement on fairness, while emphasizing more on the scheduling efficiency for higher utilization of hardware resources. Two important research problems therefore arise. First, how can the requirement of fairness-efficiency tradeoff be appropriately expressed and specified? Further, given the specified fairness-efficiency tradeoff, how can we design a practical scheduling algorithm to implement it in a real-world system?

As an initial step, in this paper, we share our thoughts and observations on the fairness-efficiency tradeoff for multi-resource packet scheduling. In particular, we present two efficiency measures for a scheduling scheme. One is the aggregated processing time that has been spent on processing the *dominant resources* — the one that requires the most processing time — of all traffic flows, while the other is simply the overall time span required to finish the services of all flows. Based on these two efficiency measures, we represent the fairness-efficiency tradeoff in two ways. The first representation is based on an idealized fluid model that strictly implements the recently proposed framework of fairness-efficiency tradeoff for multi-resource allocations [11]. This representation, though rigorous in theory, is very complicated to implement in practice using a packet-by-packet scheduling scheme. Our second representation, on the other hand, is more of a heuristic, yet is more friendly to implement in practice. We also discuss some insights on how this tradeoff can be realized using packet-by-packet queueing schemes.

II. The Fairness and Efficiency Measures

Before we discuss the tradeoff, we shall clarify the fairness and efficiency measures for a multi-resource schedule. We start off by briefly reviewing the fairness measure that has been adopted in [4], [10].

A. The Fairness Measure

The fairness of a queueing scheme is usually defined as a set of highly desired scheduling properties [4], [10]. The essential property of fair queueing is to offer *predictable service isolation* for all traffic flows. In the multi-resource setting, this means that for each flow, the received service is *at least* at the level when every resource is *equally allocated* among all flows. Further, such a service isolation should not be compromised by strategic behaviours. This requires the *truthfulness* of a schedule, in which no flow can receive better service (*i.e.*, finish faster) by artificially inflating the amount of resources it does not need.

It has been shown in [10] that by strictly implementing Dominant Resource Fairness (DRF) [5] at all times, a queueing scheme offers both predictable service isolation and truthfulness, and is therefore considered fair. That is, a fair queueing algorithm should allocate equal processing time on the *dominant resources* of all backlogged flows. The dominant resource is defined as the one that requires the most processing time among all resources. For example, in Fig. 1, a packet of Flow 1 requires $\langle 1 \text{ CPU time}, 2 \text{ Transmission time} \rangle$, so link bandwidth is its dominant resource. A packet of Flow 2, on the other hand, requires $\langle 4 \text{ CPU time}, 1 \text{ Transmission time} \rangle$, so CPU is the dominant resource. In Fig 1a, up to time 8, Flow 1 receives 6 time units to process its dominant resource (link bandwidth), while Flow 2 receives 4 time units for its dominant resource (CPU).

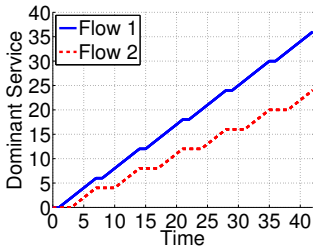
Intuitively, the fairer a queueing algorithm, the more evenly the processing time it allocates on the dominant resources of traffic flows. We therefore measure the fairness of a schedule by bounding the gap of the processing time received on the dominant resources between two backlogged flows. The resulting measure is referred to as the *Relative Fairness Bound* (RFB) [10].

Definition 1 (Relative Fairness Bound): For any packet arrival process, let $T_i(t_1, t_2)$ be the aggregate service (processing time) flow i receives on its dominant resource in the time interval (t_1, t_2). $T_i(t_1, t_2)$ is referred to as the *dominant service* flow i receives in (t_1, t_2). Let $\mathcal{B}(t_1, t_2)$ be the set of flows that are backlogged in (t_1, t_2). We define the Relative Fairness Bound (RFB) as[1]

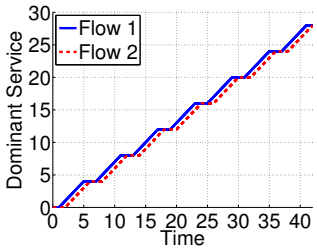
$$R = \sup_{t_1,t_2;i,j \in \mathcal{B}(t_1,t_2)} |T_i(t_1, t_2) - T_j(t_1, t_2)| \ , \qquad (1)$$

RFB measures the degree to which the DRF allocation is violated. The smaller the measure is, the fairer the scheduling algorithm will be. As an example, Fig. 2a depicts the dominant services received by both Flow 1 and Flow 2 (*i.e.*, $T_1(0, t)$ and $T_2(0, t)$, respectively), under the schedule of Fig. 1a. As we have mentioned in the introduction, this schedule, though highly efficient, allocates too many resources to Flow 1, and is unfair for Flow 2. As time goes by, the service gap between the two flows will become larger and larger, eventually leading to an unbounded RFB. In comparison, the schedule of Fig. 1b is much fairer. Fig. 2b illustrates the dominant services the schedule allocates to the two flows. We see that both flows receive almost the same services over time, with the service gap bounded below 1 time unit in all time intervals.

[1]For simplicity, we assume that all flows are of equal weight. A more general definition with different flow weights is given in [10].



(a) Services recevied in Fig. 1a.



(b) Services recevied in Fig. 1b.

Fig. 2. Illustration of the accumulated services received by two flows on their dominant resources, under the schedules of Figs. 1a and 1b.

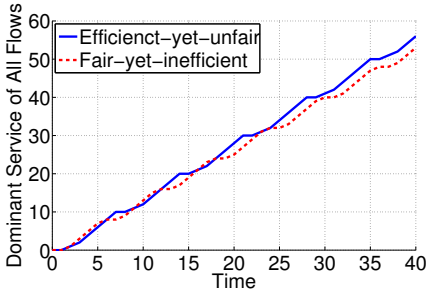


Fig. 3. Aggregate dominant services received by all flows under the schedules of Figs. 1a (referred to as Efficient-yet-unfair) and 1b (referred to as Fair-yet-inefficient), respectively.

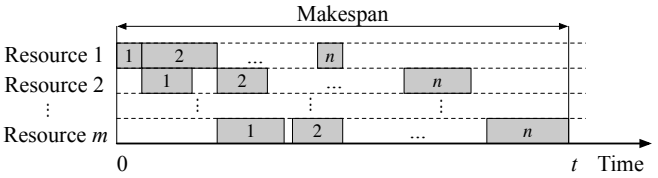


Fig. 4. Illustration of a makespan of a schedule serving n packets.

B. The Efficiency Measure

Besides fairness, scheduling efficiency is another important concern for a queueing scheme. When there is only a single resource to schedule, the efficiency is simply the utilization of that resource. In this case, maximizing the efficiency of a queueing scheme is trivially achieved by requiring the scheme to be *work conserving*. That is, whenever there are flows that are backlogged in the system, the resource must be fully utilized to serve them.

We can extend similar requirements to the multi-resource setting. In particular, a multi-resource queueing scheme is work conserving if whenever there is a flow that is backlogged in the system, then at least one resource will be fully utilized [10]. However, having this requirement will not lead to the optimal efficiency. As a counterexample, the schedule shown in Fig. 1b is work conserving. Yet it is clearly less efficient than that in Fig. 1a.

The reason that the efficiency measure becomes much more complicated in the multi-resource setting is due to the heterogeneous resource requirements of traffic flows. In this case, the resulting utilizations of different resources may differ significantly. After all, which one should be used as the efficiency measure? We explore two alternatives in the following.

Our first idea is inspired by the fairness measure. When measuring the fairness of a queueing scheme, we compare the services that a pair of flows receive on their dominant resources, while those on non-dominant resources are ignored. Based on this insight, we measure the efficiency as the aggregate services received on the dominant resources of all flows.

Definition 2 (Aggregate Dominant Service): Given an arbitrary packet arrival process, the efficiency of a schedule up to time t is measured as the *aggregate dominant services* (*i.e.,* processing time) that all flows receive, *i.e.,*

$$E_t = \sum_i T_i(0,t) \ , \tag{2}$$

where $T_i(0,t)$ is the dominant service flow i receives in $(0,t)$.

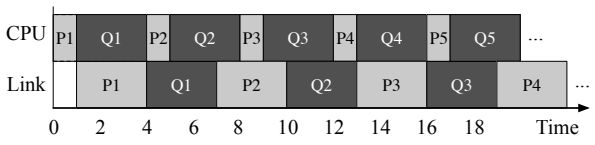
Intuitively, the more the aggregate dominant services received, the higher the efficiency of a scheduling scheme. As an example, Fig. 3 depicts the aggregate dominant services of the schedules shown in Figs. 1a and 1b. We see that though the two schedules exhibit similar efficiencies at the beginning, as time goes by, the schedule of Fig. 1a becomes more efficient than that of Fig. 1b. In fact, in Fig. 1a, up to sufficiently long time t, Flow 1 receives roughly $6t/7$ dominant services on link bandwidth, while Flow 2 receives $4t/7$ dominant services on CPU. The aggregate dominant service is then $10t/7$. In comparison, in Fig. 1b, both flows receive roughly the same services of $2t/3$ on their dominant resources, leading to an aggregate dominant service of $4t/3$. As a result, the schedule of Fig. 1a is more efficient than that in Fig. 1b (*i.e.,* $10t/7 > 4t/3$), and their efficiency gap $(10/7-4/3)t$ linearly increases with time t.

Another way to measure the efficiency is more direct. We compute the time span required to finish all services of traffic flows, and use it as the efficiency measure.

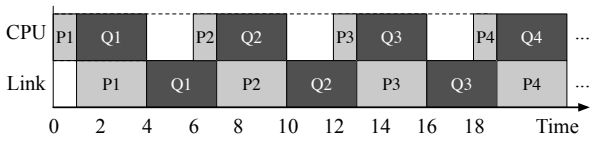
Definition 3 (Makespan): For any packet arrival process, the efficiency of a scheduling algorithm is measured as the *makespan*, *i.e.,* the total time that is required to completely process all packets.

Fig. 4 depicts the makespan of a schedule serving n packets on m resources. Intuitively, the smaller the makespan, the more efficient the schedule.

Though both aggregate dominant service (Definition 2) and makespan (Definition 3) reflect the scheduling efficiency and can be used as the efficiency measure, they are not equivalent. For example, suppose there are two traffic flows both sending 6 packets. Packet of Flow 1 requires $\langle 1$ CPU time, 3 Transmission time$\rangle$, while packet of Flow 2 requires $\langle 3$ CPU time, $3-\epsilon$ Transmission time$\rangle$, where $\epsilon > 0$ is arbitrarily small. Now consider two schedules shown in Figs. 5a and 5b, respectively. We see that the makespans of the two schedules are exactly the same. As a result, the two schemes are considered equally efficient when the makespan is used as the efficiency measure. However, as shown in Fig. 6, if the efficiency is measured by the aggregate dominant



(a) Scheduling without waiting on CPU for Flow 2.



(b) Scheduling with waiting on CPU for Flow 2.

Fig. 5. Two schedules, with and without waiting on CPU for Flow 2.

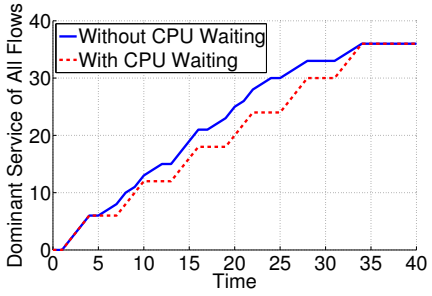


Fig. 6. Under the efficiency measure of aggregate dominant services, the scheduling without waiting on CPU is more efficient than that waiting on CPU.

services, then scheduling without CPU waiting (Fig. 5a) is more efficient than that with CPU waiting (Fig. 5b) before time 34, after which, both schemes are equally efficient.

Despite the difference of the two efficiency measures, the fairness-efficiency tradeoffs generally exist for multi-resource fair queueing. The key question is: how can such tradeoffs be expressed as a design objective of a scheduling algorithm? We investigate this problem in the next section.

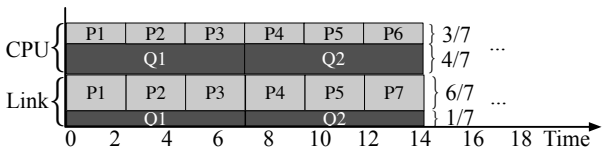
III. The Fairness-Efficiency Tradeoff

In this section, we present two ways to represent the fairness-efficiency tradeoff for a scheduling algorithm. The first representation adopts the aggregate dominant service as the efficiency measure, and is more rigorous in theory. However, it is complicated to implement in real systems. The second representation, on the other hand, uses the makespan as the efficiency measure. This representation, though heuristic, is more friendly to implement with packet-based scheduling.

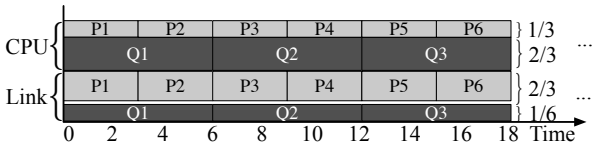
A. Tradeoff Representation Using a Unifying Framework

Our first representation is based on the recent work of [11], where a unifying framework is proposed to specify the fairness-efficiency tradeoff for multi-resource allocation. Though the framework is designed for sharing multiple resources *in space*, we show that via the idealized *multi-resource fluid flow model* [10], it can also be applied to resource scheduling, where resources are multiplexed *in time*.

Multi-Resource Fluid Model: In the multi-resources fluid model, flows are assumed to be served in *arbitrarily small*



(a) The fluid version of the schedule shown in Fig. 1a.



(b) The fluid version of the schedule shown in Fig. 1b.

Fig. 7. The fluid version of the two schedules shown in Figs. 1a and 1b.

increments. That is to say, a packet can be infinitely divided and processed partially on *every resource*. Equivalently, this implies that multiple flows can be served in parallel, each receiving fractional resources that are consumed at the same time. For example, Fig. 7 depicts the fluid versions of the two packet-based schedules shown in Fig. 1. The schedule of Fig. 1a allocates $\langle 3/7 \text{ CPU}, 6/7 \text{ Link}\rangle$ to Flow 1 and $\langle 4/7 \text{ CPU}, 1/7 \text{ Link}\rangle$ to Flow 2. The corresponding fluid schedule (Fig. 7a) allocates the same amount of resources to both flows as Fig. 1a does, except that all resources are processed simultaneously. We see that the fluid schedule retains the same fairness-efficiency tradeoff as the packet-by-packet alternative: the schedule of Fig. 7a has higher resource utilization than that of Fig. 7b, but the latter is fairer than the former.

The Fairness-Efficiency Tradeoffs: With the fluid model, a resource scheduling problem can be translated into a resource allocation problem, in which the framework of fairness-efficiency tradeoff proposed in [11] can be applied. In particular, let $\mathcal{B}(t)$ be the set of flows that are backlogged at time t. Let x_i^t be the share (fraction) of dominant resource allocated to flow i at time t, and $\mathbf{x}^t = \langle x_1^t, \ldots, x_n^t\rangle$ the allocation vector for all n flows. We express the fairness-efficiency tradeoff using the following measure originally defined in [11]:

$$f_{\beta,\lambda}(\mathbf{x}^t) = \text{sgn}(1-\beta)\left(\sum_{i\in\mathcal{B}(t)}\left(\frac{x_i^t}{\sum_{j\in\mathcal{B}(t)} x_j^t}\right)^{1-\beta}\right)^{\frac{1}{\beta}}\left(\sum_{i\in\mathcal{B}(t)} x_i^t\right)^{\lambda}. \quad (3)$$

Here, β and λ are *pre-specified parameters* representing the emphasis on fairness and efficiency, respectively. As mentioned in [11], the measure defined by (3) is divided into two components, one representing fairness and another efficiency. The term

$$\left(\sum_{i\in\mathcal{B}(t)} x_i^t\right)^{\lambda}$$

represents efficiency and is the dominant services received by all flows at time t. The remainder of (3) is parameterized by β and represents the fairness of allocation. Generally speaking, a larger β emphasizes more on fairness, while a larger λ places efficiency at a higher priority. In extreme cases, by taking $\beta = 1$, the fairness term reduces to a constant, and efficiency is the only objective to optimize. On the other hand, as $\beta \to \infty$ and $\lambda = \frac{1-\beta}{\beta}$, the measure $f_{\beta,\lambda}$ reduces to max-min fairness on the dominant shares, in which the notion of Dominant Resource Fairness (DRF) [4] is the only objective to pursue.

With the fairness-efficiency tradeoff specified as (3), we make resource allocation decisions $\mathbf{x}^t$ at all times t, so that the tradeoff measure (3) is always maximized. Specifically, let $\tau_{i,r}$ be flow i's packet processing time required on resource $r = 1, 2, \ldots, m$. Let

$$\gamma_i = \arg\max_r \{\tau_{i,r}\} \tag{4}$$

be the dominant resource of flow i. Also, let

$$\bar{\tau}_{i,r} = \frac{\tau_{i,r}}{\tau_{i,\gamma_i}}, \quad r = 1, 2, \ldots, m, \tag{5}$$

be the *normalized processing time* on resource r. Then $x_i^t \bar{\tau}_{i,r}$ is the share of resource r allocated to flow i at time t. We solve the following optimization problem to determine the dominant resource allocation $\mathbf{x}^t$ for each flow at time t:

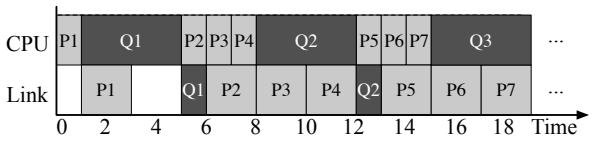
$$\begin{aligned} \max_{\mathbf{x}^t} \quad & f_{\beta,\lambda}(\mathbf{x}^t) \\ \text{s.t.} \quad & \sum_{i \in \mathcal{B}(t)} x_i^t \bar{\tau}_{i,r} \le 1, \quad r = 1, 2, \ldots, m, \end{aligned} \tag{6}$$

where the constraints ensure that the allocations will not exceed the resource capacities.

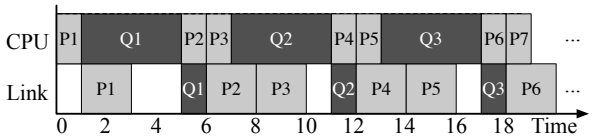
By solving (6), we obtain an idealized fluid schedule that strictly implements the specified fairness-efficiency tradeoff at all times. For example, the schedule shown in Fig. 7a (resp., 7b) is the resulting fluid schedule when efficiency (resp., fairness) is the only objective to optimize.

Packet-by-Packet Scheduling: Though the idealized fluid schedule cannot be implemented as packets are not scheduled as entities, it can be approximated by packet-by-packet queueing algorithms. Similar to how GPS [6] is approximated by fair queueing such as WFQ [6] and PGPS [7], we can maintain fluid schedule as a referencing system in the background. Whenever there is a scheduling opportunity, the packet that finishes its service the earliest in the referencing system is scheduled. For example, Fig. 8 shows the resulting packet-by-packet approximations to the two fluid schedules depicted in Fig. 7. It is easy to verify that both approximations closely track the progress of the fluid ideals. More complicated approximation techniques, such as those adopted by WF^{2}Q [12], can also be applied in a similar way.

However, we emphasize that the packet-by-packet queueing algorithm described above may be very complicated to implement, mainly due to the requirement of maintaining the referencing fluid schedule: Whenever a flow changes its state from idle to busy or from busy to idle, we have to solve the optimization problem (6), which is generally non-convex and incurs high computational complexity. To avoid this complexity, packet-by-packet scheduling should track the progress of a fluid scheme without actually emulating it. Though this can be achieved in single-resource fair queueing, *e.g.,* [13], [14], it remains open how this can be done when multiple resources are to be scheduled.



(a) Packet-by-packet approximation to the schedule of Fig. 7a.



(b) Packet-by-packet approximation to the schedule of Fig. 7b.

Fig. 8. The packet-by-packet approximations to the two fluid schedules shown in Fig. 7.

B. Tradeoff Representation as Efficiency Optimization under Fairness Constraints

The tradeoff representation presented above unifies the two requirements on both fairness and efficiency into a single design objective (6), which turns out to be complicated to implement. We now consider another tradeoff representation that takes the fairness requirement as a design constraint, under which we optimize efficiency. In particular, the fairness requirement is expressed as the RFB defined in (1), and is pre-specified by the network operator. Efficiency is measured as the scheduling makespan. The design objective is to schedule packets in a way such that they can be processed as quickly as possible, as long as the specified fairness requirement RFB is not violated. The key question here is how packets can be scheduled "as quickly as possible".

To answer this question, we derive some insights from existing solutions in the literature that consider the extreme case where efficiency is the only objective to optimize. That is, given a sequence of packets that have already arrived, how should they be scheduled to minimize the overall makespan? This is equivalent to a *multi-stage flow shop problem* [15]. In the flow shop problem, the equivalent of a packet is a job, while the equivalent of a resource is a machine. A sequence of jobs have to undergo m-stage operations on m machines, where Operation 1 must be done first on Machine 1, followed by Operation 2 on Machine 2, and so on. Different operations of a job may require different processing times. No two operations can be carried out on the same machine in parallel. The objective is to minimize the job makespan by deciding an optimal scheduling order.

When there are two resources ($m = 2$), the flow shop problem can be *optimally* solved within linear time using *Johnson's algorithm* [15] as follows. Partition the packets

into two sets, such that Set I contains those whose dominant resource is the second resource, while Set II contains all the others. The packets in Set I are scheduled first in increasing order of the processing time required on the first resource. The packets in Set II follow in decreasing order of the processing time on the second resource. Ties may be broken arbitrarily.

However, Johnson's algorithm is no longer optimal when there are more than two resources ($m \geq 3$). In fact, the flow shop problem is shown to be NP-hard in this case [15]. Despite its hardness, the intuition of Johnson's algorithm may still be applied as a good heuristic [15]. Specifically, we divide all m resources into two resource pools, with Pool I containing the first $\lfloor m/2 \rfloor$ resources and Pool II the rest. Packets are also divided into two sets. Set I contains all packets whose dominant resources fall into Pool II, while Set II contains all the others. Packets in Set I are always scheduled first, followed by packets in Set II. Now we have divided a scheduling problem with m resources into two sub-problems each with roughly $m/2$ resources. For each sub-problem, we repeat the *divide-and-conquer strategy* above, until the number of resources concerned is no more than 2, where Johnson's algorithm can be directly applied.

With Johnson's heuristic for efficiency optimization, we may expect a packet-by-packet scheduling algorithm to work as follows. It keeps track of the dominant services allocated to every traffic flow. As long as the service gap does not exceed the specified fairness requirement, *i.e.,* RFB in (1), we schedule packets using Johnson's heuristic for the sake of higher efficiency. Once the service gap approaches the specified fairness bound (*i.e.,* exceeds a threshold), the flow that receives the least service on its dominant resource will have the highest priority to be served until the gap falls below the threshold, after which efficiency will become the primary concern and the whole process repeats. Compared with the tradeoff representation defined as a unifying framework in Sec. III-A, the representation above is heuristic, yet is more friendly to implement using a packet-by-packet scheduling algorithm.

Though we have briefly outlined the two tradeoff representations and compared their pros and cons, there remains a long way away from a concrete implementation that offers flexible tradeoff between fairness and efficiency in multi-resource scheduling. We share our views about the future work in the final section.

IV. Concluding Remarks and Future Work

Middleboxes are ubiquitous in today's datacenter networks. They apply complex network functions that require multiple hardware resources, *e.g.,* CPU, memory bandwidth, link bandwidth, *etc*. Depending on the network functions that packets go through, different traffic flows may require vastly different amounts of various resources. Unlike single-resource packet scheduling, we show that there exists a tradeoff between fairness and efficiency in multi-resource packet scheduling. We call attention to two important research problems: (1) how can the fairness-efficiency tradeoff be expressed, and (2) how can a queueing algorithm be designed to implement the specified tradeoff. As an initial step, we present two tradeoff representations. The first is based on a unifying framework in which both the fairness and the efficiency concerns are expressed as a single optimization objective. This representation, though rigorous in theory, is hard to implement directly in practice. We also give another tradeoff representation that can be heuristically implemented with packet-by-packet scheduling, in which efficiency is optimized under some specified fairness constraint.

We believe that both tradeoff representations deserve further investigation in the future. For the first representation, the key problem is how to closely track the fluid schedule using a packet-by-packet scheme with low complexity. Similar challenges are also found in the traditional fair queueing literature, where various designing insights and techniques have been proposed to approximate GPS. Whether and how these insights and techniques can be applied to the multi-resource scenario is an interesting question to answer. For the second tradeoff representation, a more concrete heuristic implementation is still on the way. So far, it remains unclear how its performance will be and to what extent the heuristic can accurately produce the specified fairness-efficiency tradeoff.

References

[1] V. Sekar, N. Egi, S. Ratnasamy, M. Reiter, and G. Shi, "Design and implementation of a consolidated middlebox architecture," in *Proc. USENIX NSDI*, 2012.

[2] J. Sherry, S. Hasan, C. Scott, A. Krishnamurthy, S. Ratnasamy, and V. Sekar, "Making middleboxes someone else's problem: Network processing as a cloud service," in *Proc. ACM SIGCOMM*, 2012.

[3] N. Egi, A. Greenhalgh, M. Handley, M. Hoerdt, F. Huici, and L. Mathy, "Towards high performance virtual routers on commodity hardware," in *Proc. ACM CoNEXT*, 2008.

[4] A. Ghodsi, V. Sekar, M. Zaharia, and I. Stoica, "Multi-resource fair queueing for packet processing," in *Proc. ACM SIGCOMM*, 2012.

[5] A. Ghodsi, M. Zaharia, B. Hindman, A. Konwinski, S. Shenker, and I. Stoica, "Dominant resource fairness: Fair allocation of multiple resource types," in *Proc. USENIX NSDI*, 2011.

[6] A. Demers, S. Keshav, and S. Shenker, "Analysis and simulation of a fair queueing algorithm," in *Proc. ACM SIGCOMM*, 1989.

[7] A. Parekh and R. Gallager, "A generalized processor sharing approach to flow control in integrated services networks: The single-node case," *IEEE/ACM Trans. Netw.*, vol. 1, no. 3, pp. 344–357, 1993.

[8] H. Zhang, "Service disciplines for guaranteed performance service in packet-switching networks," *Proc. IEEE*, vol. 83, no. 10, pp. 1374–1396, 1995.

[9] S. Keshav, *An Engineering Approach to Computer Networking: ATM Networks, The Internet and Telephone Network*. Addison-Wesley Longman Publishing Co., Inc., 1997.

[10] W. Wang, B. Liang, and B. Li, "Multi-resource generalized processor sharing for packet processing," University of Toronto, Tech. Rep., 2013. [Online]. Available: http://iqua.ece.toronto.edu/~bli/papers/mrgps.pdf

[11] C. Joe-Wong, S. Sen, T. Lan, and M. Chiang, "Multi-resource allocation: Fairness-efficiency tradeoffs in a unifying framework," in *Proc. IEEE INFOCOM*, 2012.

[12] J. Bennett and H. Zhang, "WF^{2}Q: Worst-case fair weighted fair queueing," in *Proc. IEEE INFOCOM*, 1996.

[13] S. Golestani, "A self-clocked fair queueing scheme for broadband applications," in *Proc. IEEE INFOCOM*, 1994.

[14] P. Goyal, H. Vin, and H. Cheng, "Start-time fair queueing: A scheduling algorithm for integrated services packet switching networks," *IEEE/ACM Trans. Netw.*, vol. 5, no. 5, pp. 690–704, 1997.

[15] M. L. Pinedo, *Scheduling: Theory, Algorithms, and Systems*. Springer, 2012.