ATFQ: A Fair and Efficient Packet Scheduling Method in Multi-Resource Environments

Jianhui Zhang, Heng Qi, Deke Guo, Keqiu Li, *Senior Member, IEEE*, Wenxin Li, and Yingwei Jin

***Abstract*—Large-scale data centers are the key infrastructures for hosting and running a variety of applications. Besides traditional L2/L3 devices, middleboxes are widely deployed in data centers and perform many important functions, e.g., the intrusion detection and firewall. Middleboxes are equipped with multiple kinds of resources, such as CPU and memory. Data flows undergoing different functions have heterogeneous processing time requirements on diverse resources. Researchers are in a dilemma as to how to provide fair service for flows and efficiently utilize those scarce resources. To address this problem, we propose a novel packet scheduling method, active time fairness queuing (ATFQ), for multi-resource environments. Prior packet scheduling methods usually focus on pursuing the fairness among flows, resulting in enormous waste of those scarce resources. ATFQ overcomes this essentially by redefining the fairness and can maximize the resource utilization with the guarantee of fairness. We conduct extensive simulations to evaluate the performance of ATFQ. The evaluation results demonstrate that flows get better service in many aspects under ATFQ. Meanwhile, the resource utilization rises up by about 10% than the traditional DRFQ, which is one of the mainstream involved methods.**

***Index Terms*—Multi-resource, packet processing, fair scheduling, efficiency.**

I. Introduction

With the growth of scale and the enhancement of functionality, data centers accommodate more and more applications. Data centers are the key infrastructures and provide reliable service for these applications. Besides traditional L2/L3 devices, such as routers and switches, middleboxes are widely deployed in data centers [1], [2]. Traditional network devices are only responsible for the basic routing and forwarding. The major advantage introduced by middleboxes is that they perform many data processing functions, e.g., intrusion detection, network address translation, and firewall. To some degree, middleboxes alleviate the pressure of servers and optimize the network environment for data centers. Meanwhile, middleboxes are usually equipped with multiple kinds of resources, e.g., CPU, memory, and link bandwidth, which are multiplexed by flows passing through them. Performing different functions induces distinct consumptions on those resources. For example, the intrusion detection performs the deep analysis on the content of packets. Hence, it bottlenecks on the CPU. If flows just need to be forwarded, the link bandwidth becomes the bottleneck more likely. What's more, processing large amount of small packets consumes more memory bandwidth in software routers [3]. It is worth noting that the amount of resources at a middlebox is very scarce, compared to that at a server. Due to the huge volume of traffic passing through middleboxes, how to provide fair service for these data flows and efficiently utilize the middlebox resources becomes a serious challenge for a packet scheduling method. Unfortunately, existing scheduling methods in a multi-resource environment focus only on the former and induce an enormous waste of resources.

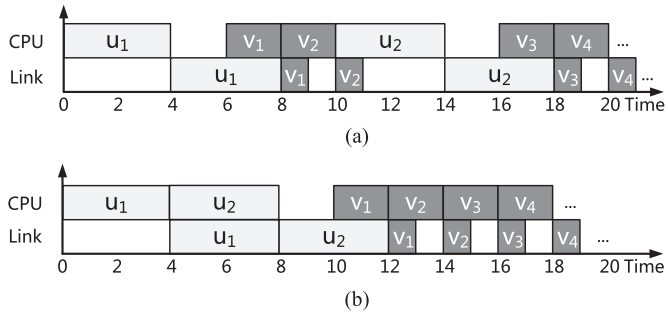


Fig. 1. Improve the efficiency by relaxing the constraint of fairness. (a) Strict DRFQ leads to the loss of efficiency. (b) Relax DRFQ to improve the efficiency.

In the setting of a single resource environment, traditional packet scheduling methods focus only on providing fair service for flows or shortening the flow completion time (FCT). They, however, concern nothing about the resource utilization. Actually, the resource will be fully utilized if such methods satisfy the work conservation, i.e., the resource should not be wasted in idle state when there exist packets needing to be processed. Similarly, in a multi-resource environment, work conservation can be defined as that at least one resource is fully utilized when there exist packets that have not yet been processed. However, different from the single resource environment, packet scheduling methods satisfying this definition exhibit different levels of resources utilizations in a multi-resource environment. To ease the presentation, we use Fig. 1 as an example. Here we assume there are only two kinds of resources, i.e., the CPU and the link,

Manuscript received March 24, 2015; revised July 2, 2015; accepted September 6, 2015. Date of publication September 9, 2015; date of current version December 17, 2015. This work was supported by the National Science Foundation for Distinguished Young Scholars of China (Grant 61225010); the National Basic Research Program (973 program) under Grant 2014CB347800; the State Key Program of National Natural Science of China (Grant 61432002); the NSFC Grant 61272417, 61300189, 61370199, 61422214, and 61173161; the Specialized Research Fund for the Doctoral Program of Higher Education (Grant 20130041110019), and the Fundamental Research Funds for the Central Universities (Grant DUT15QY20). This work was done during internship at National University of Defense Technology. The associate editor coordinating the review of this paper and approving it for publication was P. Bellavista.

J. Zhang, H. Qi, K. Li, and W. Li are with the School of Computer Science and Technology, Dalian University of Technology, Dalian 116023, China (e-mail: zhangjh@mail.dlut.edu.cn; hengqi@dlut.edu.cn; likeqiu@gmail.com; liwenxin@mail.dlut.edu.cn).

D. Guo is with the School of Information System and Management, National University of Defense Technology, Changsha 410073, China (e-mail: guodeke@gmail.com).

Y. Jin is with the School of Management, Dalian University of Technology, Dalian 116023, China (e-mail: jinyw67@dlut.edu.cn).

Digital Object Identifier 10.1109/TNSM.2015.2477517

in the system and they are multiplexed by flow U and flow V. Each packet of flow U, denoted by $u_1, u_2, \ldots$, needs 4 time units respectively on the CPU and the link. Meanwhile, each packet of flow V, denoted by $v_1, v_2, \ldots$, needs 2 time units on the CPU and 1 time unit on the link. Furthermore, packets can be transferred to the link only after being processed on the CPU, which agrees with the practical situation.

DRFQ [4] is a timestamp based scheduling method and generalizes the max-min fairness into a multi-resource environment. As in DRFQ, different flows should receive equal processing time on their respective dominant resources. The dominant resource of a flow indicates the resource with the largest processing time for that flow among all the resources. In this way, such two flows will enter a scheduling cycle, where one packet of flow U will be processed after every two packets of flow V, as illustrated in Fig. 1(a). In the long term, the utilizations of the CPU and the link are 80% and 60%, respectively. Taking the resource utilization into consideration, we can loosen the constraint of the fairness a little bit and equalize different flows' processing time on their respective dominant resources in a relaxed way. For example, continuously schedule 4 packets of flow V after 2 sequential packets of flow U, as illustrated in Fig. 1(b). In this way, the utilizations of the CPU and the link rise up to 88.9% and 66.7%, respectively. Accordingly, we can reasonably conclude that work conservation cannot guarantee high resource utilization in a multi-resource environment. What's more, strictly pursuing the fairness among flows always leads to serious waste of resources.

In a multi-resource environment, the resource utilization influences the quality of service for flows in many aspects. Higher resource utilization means that more packets or flows can be processed within the same time interval, compared to a scheduling scenario of low resource utilization. Accordingly, each flow will get a shorter completion time, which is usually the main concern for flows. So, besides the fairness, the resource utilization is another essential metric for evaluating the performance of the scheduling method, especially in a multi-resource environment. Prior packet scheduling methods in a multi-resource environment, e.g., DRFQ [4], DRGPS [5], and MR^3 [6], only strive to provide fair service for flows, but all of them suffer considerable loss of the resource utilization. As illustrated in Fig. 1(b), it's feasible for a packet scheduling method to achieve high resource utilization while providing fair service for flows.

However, it's a challenge to achieve the fairness among flows and high resource utilization simultaneously. Different from traditional packet scheduling methods in a single-resource environment, the definition of fairness is still ambiguous in a multi-resource environment, not to mention how to promote the resource utilization. In this paper, we propose ATFQ (active time fairness queueing), a novel packet scheduling method that takes the fairness and the resource utilization as the joint scheduling goal. For avoiding the drawback of DRFQ, we redefine the fairness in a multi-resource environment as ATF (active time fairness) to provide fair service for flows. Meanwhile, by using an efficient scheduling algorithm, we can maximize the resource utilization with the guarantee of the fairness. We conduct extensive evaluations to verify the performance of our proposal. The evaluation results demonstrate that ATFQ ensures the fairness among flows and considerably improves the utilizations of all kinds of resources. Consequently, flows achieve better service in many aspects under ATFQ, compared to other scheduling methods that only focus on providing fair service isolation for flows.

The main contributions of this paper are summarized as follows:

1) For avoiding the drawback of DRFQ, we redefine the fairness as ATF to ensure fair service for flows in a multi-resource environment.
2) Besides the fairness, we integrate the resource utilization into the scheduling objective and design a packet scheduling algorithm to maximize the resource utilization in a multi-resource environment.
3) We explore the tradeoff between the fairness and the promotion of the resource utilization. Relaxing the constraint of the fairness a little bit can increase the resource utilization by different degrees.
4) We conduct extensive evaluations to verify the promotion for flows in many aspects, benefiting from high resource utilization.

The rest of this paper is organized as follows. In Section II we discuss the properties and drawbacks of prior packet scheduling methods. Section III presents a new definition of fairness to ensure fair service for flows in a multi-resource environment. Section IV designs an efficient packet scheduling algorithm to maximize the resource utilization. In Section V, we conduct extensive evaluations to verify the performance of ATFQ. Finally Section VI concludes our work.

II. Preliminaries

We start with some packet scheduling methods for a single resource and multiple resources respectively. We then discuss the importance of the resource utilization.

A. Packet Scheduling for a Single Resource

In the network field, packet scheduling is always an important problem and attracts lots of attentions. Traditional scheduling methods are designed for a single resource, e.g., the link bandwidth and the buffer queue. Such methods are usually designed for specified objectives and can be correspondingly categorized into two families.

1) Fairness Oriented: Flows should be treated equally when multiplexing the resource. Fairness based scheduling methods strive to make fair service isolation for flows, according to the specified definition of the fairness, e.g., the max-min fairness and the proportionate fairness [7]. Flow processing time or the received service time is usually taken as a metric to allocate scheduling opportunities for flows. Traditionally, data packet is processed as an entirety and one resource can only process one packet at a time. Scheduling packets in any sequence will inevitably lead to a service difference among flows. Generalized processor sharing (GPS) [8] has been proposed to eliminate this difference by assuming that packets can be divided infinitely and the resource can be simultaneously multiplexed by multiple packets. GPS can be seen as a benchmark and

a lot of scheduling methods try to approximate it, e.g., FQ (fair queueing) and WFQ (weighted fair queueing) [9]. WF^2Q (worst-case fair weighted fair queueing) [10], SCFQ (self-clocked fair queueing) [11] and SFQ (start-time fair queueing) [12] refine the performance of fair queueing in some special scheduling scenarios.

2) Focusing on the Flow Completion Time: FCT associates with many factors, e.g., the transport control protocol, the flow scheduling algorithm, and the network load. Many schemes have been proposed to shorten it through different ways.

RCP (rate control protocol) [13] assigns the same rate to the flows passing through each router, so as to emulate processor sharing at routers. Subsequently, literatures [14], [15] adopt such a technique to shorten FCT. In data centers, throughput sensitive flows coexist with latency sensitive flows. The latter suffers great queueing delay when the former occupies most of the buffer space. For alleviating this situation, DCTCP [16] leverages explicit congestion notification (ECN) to maintain low queue length in switches. Like DCTCP, L^2DCT [17] adopts ECN to estimate the extent of network congestion. Moreover, it modulates the congestion window size based on the amount of traffic that a flow has sent, so as to reduce the completion times of short flows. PDQ [18] schedules flows in a preemptive and distributed way and prefers flows with urgent criticality. The flow criticality can be estimated based on the flow size or the flow deadline. Thus, PDQ can implement SJF or EDF in the whole network to shorten the FCT or guarantee the flow deadline. In data centers, the long-tailed distribution of FCT extremely impacts the performance of some web applications. Based on such an observation, Detail [19] implements a cross-layer network stack to provide reliable transmission for flows. Meanwhile, it shortens the tail of completion times of latency-sensitive flows by prioritizing such flows. By contrast, pFabric [20] achieves near optimal FCT through some simple schemes. It decouples flow scheduling from rate control. Packets are scheduled based on their priorities in switches. All flows start at line-rate and reduce their sending rate when persistent packet loss arises. The above mentioned scheduling schemes usually rely on custom switches and modifications on protocol. In addition, related flow information should be collected in advance. By contrast, PIAS [21] leverages a multiple level feedback queue to emulate SJF without detecting flow sizes beforehand.

On the other hand, the unbalanced distribution of network load also disturbs FCT. Multipath TCP (MPTCP) [22] sufficiently improves network throughput by spreading data across several subflows [23]. Meanwhile, FCT can be reduced by moving traffic away from congested links [24]. Typically, RepFlow [25] replicates each short flow and transmits the replicated and original flows on different paths. Thus, congested links with long queueing delay can be avoided and FCT can be correspondingly reduced. Some bandwidth allocation schemes [26], [27] have also been proposed to provide predictable bandwidth for flows, so as to shorten the FCT.

As for soft real-time applications, e.g., web search, the completion time of each flow should not be later than its deadline. Otherwise, such flows will not be added into the final response to users. For satisfying flows' deadline requirements, more detailed information is needed when making scheduling decisions. D^3 [28] adjusts the flow sending rate based on the remaining flow size and the deadline. Similarly, D^2TCP [29] implements a congestion avoidance algorithm to modify the congestion window. Flows are treated differently and those with urgent deadlines are preferred.

B. Packet Scheduling for Multiple Resources

The flow scheduling problem in a multi-resource environment can also be seen as a resource allocation problem in time dimension. However, the concept of the fairness is ambiguous in a multi-resource environment. Simply, we can extend the related definitions of fairness in a single resource environment to a multi-resource environment. Equally partitioning all resources is the simplest manner. Bottleneck fairness [4] allocates the most required resource equally to users. For the asset fairness, different resources can interchange with each other and each user will finally get the same worth of resources. Recently, a resource allocation scheme, dominant resource fairness (DRF) [30], generalizes the concept of max-min fairness into a multi-resource environment. In the setting of DRF, each user gets the same proportion of its dominant resource, which indicates the resource with the maximal proportion of occupancy among all resources. DRF possesses many attractive properties and some corresponding packet scheduling methods have been proposed.

Resorting to the fluid model, where packets can be infinitely divided, DRFQ [4] equalizes different flows' processing time on their respective dominant resources. DRFQ is a timestamp based scheduling method and needs a large amount of computation before making scheduling decision. To lower down the implementation complexity of DRFQ, MR^3 [6] uses a round robin algorithm to achieve DRF and makes scheduling decision in $O(1)$. However, flows with higher priorities experience longer inter-packet delay in MR^3. GMR^3 [31] improves MR^3 by dividing flows into groups based on their weights. The group with the earliest timestamp will be chosen and the flows within this group will be scheduled in a round-robin fashion. DRGPS [5] generalizes the concept of GPS into a multi-resource environment and realizes the strict DRF for flows. However, per-flow buffer leads enormous memory cost when the number of flows is large [32]. Myopia [32] leverages an improved CM Sketch to detect elephant flows and allocates per-flow buffer to such flows. Mice flows are stored in a FIFO queue and share the buffer together. Unfortunately, for achieving the fairness, these scheduling methods all suffer considerable loss of the resource utilization.

C. Packet Scheduling With Fairness and Efficiency

As for the packet scheduling problem, the most important difference between a single resource environment and a multi-resource environment is the resource utilization. For a single resource, it will be fully utilized if it keeps on processing packets. In a multi-resource environment, such as middleboxes, work conservation cannot ensure high resource utilization. Resources should be efficiently utilized to provide better service for flows. With high resource utilization, more flows can be processed

within the same time interval and flows will be finished faster. Thus, besides the fairness, the resource utilization should also be taken as an important scheduling objective. Although literature [33] notices the tradeoff between fairness and efficiency in a multi-resource environment, the existing packet scheduling methods focus only on the fairness and neglect the importance of the resource utilization.

The resource utilization can be seen as a metric for evaluating the efficiency of a scheduling method. Generally, the fairness and the efficiency appear to be at odds. Realizing the strict fairness always means more fine-grained control. Conversely, more control overhead will lower down the scheduling efficiency. It's not a trivial job to make the best of both worlds. In this paper, we try to achieve high resource utilization with the guarantee of fairness for flows. We design a new measurement of fairness to make service isolation for flows. Meanwhile, our scheduling algorithm can maximize the resource utilization. Benefiting from this, flows get better service in many aspects, including the completion time and the inter-packet delay.

III. The Schedule Motivation

We start with the new concept of the active time fairness. Then we propose the methods to measure the active time and the efficiency, respectively. Finally, we discuss how to improve the efficiency while ensuring the fairness.

A. Active Time Fairness

Before further discussion, we need to clarify the meaning of the fairness for the packet scheduling problem in a multi-resource environment. Inspired by DRF [30], we present some attractive properties that should be satisfied by fairness driven packet scheduling methods as follows:

1) *Fair rate isolation*: when n flows share the multiple resources, the rate of each flow should be $\frac{1}{n}$ of α, where α is the rate when it monopolizes all the resources.
2) *Inter-packet delay*: the start time of each flow's adjacent packets should be bounded within an acceptable range.
3) *The difference of flows' received service time*: the difference of flows' received service time should be bounded within an acceptable range.
4) *Strategy-proofness*: flows cannot get more service by lying about their required processing time on resources.
5) *Starvation-proofness*: flows should not wait for a very long time before being processed.

DRFQ takes flows' processing time on their respective dominant resources as the indicator of their received service time. DRFQ follows the principle of the max-min fairness when making scheduling decisions. However, as for a flow, the sizes of its packets are changeable. This leads to instable processing time requirements on diverse resources. Consequently, the dominant resource for a flow may shift from one resource to another. We interpret this phenomenon in Fig. 2.

At the beginning, flow W sends 5 packets, w_1–w_5, and each packet requires 2 time units on the CPU and 1 time unit on the link. Then it sends 6 packets, w_6–w_{11}, and each packet requires 1 time unit on the CPU and 2 time units on the link. Obviously, as for this flow, its dominant resource gradually shifts from the CPU to the link. Before the completion of the shift, the CPU is still the dominant resource and the sum of processing time on the CPU will increase slowly. That is, its packets will get smaller timestamps than usual, where the dominant resource does not shift. Consequently, flow W gets more scheduling opportunities than other flows. This phenomenon deteriorates seriously when it has extremely different processing time on diverse resources. Consequently, other flows may be starved before the completion of the flow, whose dominant resource changes. For avoiding this drawback of DRFQ, we redefine the fairness in a multi-resource environment as the active time fairness. Before that, we introduce the active time.

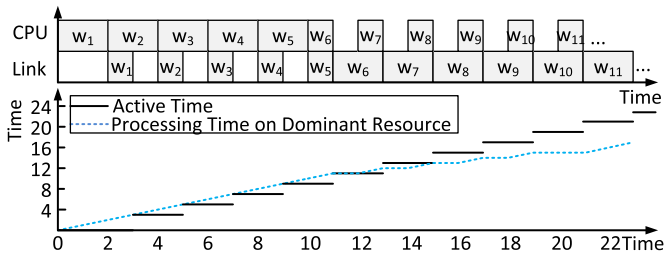


Fig. 2. Shift of the dominant resource.

*Definition 1 (**Active Time**):* A flow's active time is the time interval from the beginning of using the first resource to the release of the last resource when it monopolizes all resources.

The active time records the flow's processing procedure, so we can reasonably take the active time as a flow's received service time. As for a flow, the difference between the processing time on its dominant resource and the active time can be distinguished, as shown in Fig. 2. If taking the active time as the timestamp of packet, the subsequent packets will not get smaller timestamp. Obviously, the active time changes steadily and is immune to the shift of dominant resource. According to Definition 1, we define the active time fairness in a multi-resource environment as follows.

*Definition 2 (**Active Time Fairness**):* Each flow should receive the equal active time when sharing all the resources with other flows.

ATF gets inspiration from the virtual clock [34] and generalizes its concept into a multi-resource environment for the packet scheduling problem.

B. Measurement of the Active Time

The setting of the multi-resource extremely complicates the measurement of the active time. Each packet experiences multiple processing phases on different resources and has different processing time on them. For making the measurement precise, we model and formalize the packet processing procedure, as illustrated in Fig. 3. Assume there are m kinds of resources, denoted by R_1, R_2, . . ., and R_m, which are numbered according to the packet processing procedure. The same color graph indicates the sequential packet processing phases on different

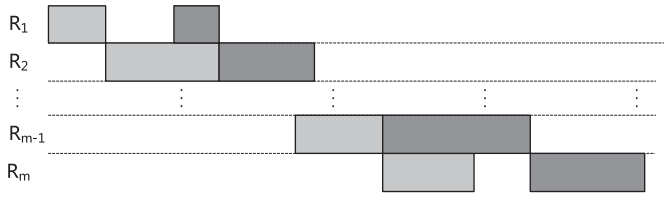


Fig. 3. Packet processing procedure.

TABLE I
NOTATIONS USED FOR TIMESTAMP COMPUTATION

Notation	Explanation
m	the number of the resources
a_i	the arrival time of flow f_i
p_i^k	the k-th packet of flow f_i
$L(p_i^k, j)$	the processing time of p_i^k on resource j
$S(p_i^k, j)$	the start time of p_i^k on resource j
$F(p_i^k, j)$	the finish time of p_i^k on resource j

resources. In summary, the packet processing procedure obeys the following rules:

1) One resource can only process one packet at any time.
2) One packet can be processed on only one resource at any time.
3) All packets should be processed in the same resource sequence.
4) The processing of a packet cannot be interrupted before its completion on all the resources.

The first and second rules make sense in most situations, except for some special resources that can process multiple packets simultaneously, e.g., the multi-core CPU. This beyond the scope of this article and will not be taken into consideration. As for the third rule, the packet processing follows a fixed sequence on all the resources and this processing sequence cannot be disturbed arbitrarily. That is, packets need to be processed on the CPU before transmitting it on the link. Buffers, if any, between adjacent resources can temporarily store the packets but interrupt the packet processing procedure. So rule 4 results from the assumption that there exist no buffers. Before moving forward, Table I summarizes some necessary notations. As for a backlogged flow f_i, its sending rate exceeds the processing rate of the system. So, packet p_i^{k+1} arrives before the completion of packet p_i^k on the first resource. When flow f_i monopolizes all the resources, the start and finish time of its packets on all the resources can be got.

According to the definition of the active time, we take $F(p_i^k, m)$ as an indicator of the active time for flow f_i when p_i^k has been completely processed on all of the resources. Suppose that p_i^k is the first packet of flow f_i, which newly arrives or just recovers from idle to busy. For synchronizing the processing of flow f_i with other flows that have already been scheduled, the timestamp of p_i^k needs to be mapped as an appropriate value. Actually, there are many ways to realize this mapping operation. For making the newly arrived flows be scheduled as soon as possible, we set $F(p_i^k, m)$ as the maximal finish time in the set of packets being processed at a_i, denote by $Q(a_i)$. Let q denote any packet belonging to $Q(a_i)$, the mapping operation can be formalized as:

$$F\left(p_i^k, m\right) = \begin{cases} \max\limits_{q \in Q(a_i)} F(q, m) & Q(a_i) \neq \phi \\ 0 & Q(a_i) = \phi \end{cases} \quad (1)$$

According to the fourth rule aforementioned, the packet will be pushed to the next resource immediately after finishing its processing on the current resource. So, the start time of p_i^k on the first resource can be calculated as:

$$S\left(p_i^k, 1\right) = F\left(p_i^k, m\right) - \sum_{b=1}^{m} L\left(p_i^k, b\right). \quad (2)$$

As for the subsequent packets of p_i^k, we get the start time of p_i^{k+1} on the first resource as:

$$S\left(p_i^{k+1}, 1\right) = S\left(p_i^k, 1\right) + L\left(p_i^k, 1\right) + \max_n \left(\sum_{j=2}^{n} L\left(p_i^k, j\right) - \sum_{j=1}^{n-1} L\left(p_i^{k+1}, j\right), 0 \right), \quad (3)$$

where $2 \leq n \leq m$. The start time of p_i^{k+1} on the other resources can be deduced as:

$$S\left(p_i^{k+1}, n\right) = F\left(p_i^{k+1}, n-1\right) = S\left(p_i^{k+1}, n-1\right) + L\left(p_i^{k+1}, n-1\right). \quad (4)$$

Finally, the finish time of p_i^{k+1} on the final resource can be got as:

$$F\left(p_i^{k+1}, m\right) = S\left(p_i^{k+1}, m\right) + L\left(p_i^{k+1}, m\right). \quad (5)$$

That is, the first arrived packet of a flow will be allocated a timestamp according to the backlogged flows in the middlebox. The timestamp of the subsequent packets will be computed based on the previous packets. Although flows multiplex all the resources with each other, their timestamp computations are conducted respectively, so as to measure their active time.

C. Measurement of the Efficiency

For a single resource, higher resource utilization means flows can be processed faster and more flows can be processed within the same time interval. Makespan, i.e., the total processing time, is usually taken as the metric to measure the efficiency of a scheduling method. A scheduling method with high efficiency will sufficiently utilize the resources to process flows quickly. So the efficiency of the scheduling method is equivalent to the resource utilization in most scenarios. In a multi-resource environment, two metrics can be adopted to measure the efficiency of a scheduling method:

1) The sum of flows' processing time on their respective dominant resources.
2) The total processing time interval for flows.

The first metric partially emphasizes the importance of the processing time on the dominant resource, violating the definition of the ATF. In addition, if we neglect the concrete packet processing on the multiple resources and see these resources as whole, higher resource utilization still leads to shorter completion time for flows. Obviously, the second metric is a more direct representation for the efficiency. So in this paper, we take the second metric as the definition of the efficiency. So far we have defined the measurements of the fairness and efficiency. The challenge turns out to be how to achieve them simultaneously. This is also the main concern of this paper.

D. Achieving the Fairness and Efficiency Simultaneously

DRFQ is fairness-driven and generalizes the max-min fairness into multi-resource environments. Thus, the difference of flows' processing time on their respective dominant resources will be restricted within the minimal range. We can achieve the strict ATF for flows by always scheduling the packet with the smallest finish time, i.e., the flow with the smallest active time. But as aforementioned, this will lead to considerable waste of the resources, which in turn influences the quality of service for flows. The constraint of the fairness is not competent in a multi-resource environment. So, we introduce the concept of the time domain, so as to realize the ATF for flows in a relaxed manner and improve the efficiency simultaneously.

A time domain, denoted by $[T1, T2)$ for $T1 < T2$, can be seen as a relatively longer time interval, compared to the packet processing time. According to the definition of the active time, the scheduler makes timestamp computation respectively for each flow with the assumption that it monopolizes all the resources. The packets within $[T1, T2)$, i.e., their finish times are no less than $T1$ and less than $T2$, will be processed in the same scheduling loop. The next scheduling loop will not start until all the packets in the current scheduling loop have been completely processed. The packets belonging to different flows have heterogeneous requirements of the processing time on diverse resources. To obey the constraint of the time domain, flows differ in the number of packets being processed in each scheduling loop. So in every scheduling loop, flows receive roughly the same active time. In the long term, flows get the equal active time and the ATF can be achieved.

For improving the efficiency of the scheduling method, all the resources should be sufficiently utilized to process flows quickly. However, flows arrive frequently and their arrivals cannot be predicted. Maximizing the efficiency of the scheduling method in a global way is unrealistic. Consequently, we focus on the total processing time interval for all the packets in the same scheduling loop and minimize it by adopting an efficient scheduling algorithm. The scheduling algorithm will be explained in Section IV-B.

IV. The Design and Implementation of ATFQ

We start with the design rationales of ATFQ and then present the efficient scheduling algorithm. We finally explore the properties of ATFQ in detail.

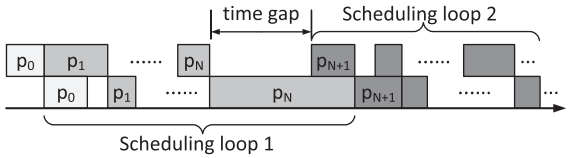


Fig. 4. Scheduling loops.

A. The Design Rationales of ATFQ

ATFQ is not a scheduling method only focusing on providing fair service for flows. Besides the fairness, it also takes the efficiency as the scheduling objective. ATFQ does not pursue the strict fairness for flows, where the difference of received service time of flows can be bounded within the minimal range. In ATFQ, the fairness among flows will be achieved in a relaxed way and the efficiency can be maximized simultaneously by adopting the time domain. Packets obeying the constraint of the time domain will be processed in the same scheduling loop. Although different flows receive roughly the same active time in every scheduling loop, in the long term, there will be an equivalent on flows' active time.

As for each scheduling loop, we use a scheduling algorithm to minimize the total processing time interval for packets in each time domain. To ease the presentation, we illustrate the scheduling loops in Fig. 4. The same color graphs indicate the packets belonging to the same scheduling loop. Although we can minimize the total processing time intervals for scheduling loop 1 and scheduling loop 2 respectively, the two time intervals overlap when processing packet p_{N+1} on the first resource. In addition, there may exist a time gap between the packets on their boundary, i.e., packets p_N and p_{N+1}. Here the time gap refers to the idle time between adjacent packets on the first resource. So, the packet scheduling sequence is not determined only based on the information of the packets in the current scheduling loop. The final scheduled packet in the previous scheduling loop, i.e., p_N, should be taken into account when selecting p_{N+1} as the first packet in the next scheduling loop. The first resource will be wasted to some extent if neglecting the time gap between adjacent scheduling loops. To avoid this drawback, we need to redefine the scheduling objective for every scheduling loop.

For N packets, $p_1, \ldots$, and p_N, in the scheduling loop 1, their processing time on the first resource, $L(p_i, 1)$ for $1 \leq i \leq N$, cannot be changed. However, the sum of the time gap between these packets is a variable associating with the specified packet scheduling sequence. Let $G(p_i)$ denote the time gap between packets p_i and p_{i-1}. For a given packet scheduling sequence in scheduling loop 1, we define $\mathbb{E}$ as:

$$\mathbb{E} = \sum_{i=1}^{N} \left(L(p_i, 1) + G(p_i)\right). \tag{6}$$

Based on this definition, the value of $\mathbb{E}$ measures the packet processing time interval on the first resource for the packets belonging to the same scheduling loop. Note that it includes the time gap after the last packet scheduled in the previous scheduling loop, but not the time gap after the last packet scheduled

in the current scheduling loop. Obviously, as for different scheduling loops, the computations of $\mathbb{E}$ have no overlap on time. So they perfectly divide the packet scheduling procedure in logic. By doing this, each scheduling loop can compute its value of $\mathbb{E}$ and try the best to minimize it, respectively.

It is worth noting that if the packets belonging to the same scheduling loop can be completely processed within the minimal time interval on the first resource and their number is large enough, the makespan also approximates to the its minimum. Here we follow this observation and take $\mathbb{E}$ as the representation of the makespan. It is reasonable and efficient, as verified in our performance evaluations. Consequently, we design a scheduling algorithm to minimize $\mathbb{E}$ for every scheduling loop, so as to maximize the efficiency.

B. Efficient Scheduling Algorithm

According to the proposed packet processing model, when $m > 2$, minimizing the total processing time interval for packets in one scheduling loop is a NP-hard problem. Approximate solutions can be got through some heuristic algorithms. In this paper, we only consider the scheduling scenario of two resources, where the optimal solution can be achieved. As an exception, assume there are m kinds of resources and N packets will be processed in one scheduling loop. In addition, the packet scheduling sequence with the minimal $\mathbb{E}$ is given. Now we add a new resource as the final resource and assume the maximal packet processing time on the $(\mathrm{m}+1)$-th resource is smaller than the minimal packet processing time on the m-th resource. Based on these assumptions, we get:

$$\begin{aligned}\sum_{j=2}^{m+1} L(p_i,j) - \sum_{j=1}^{m} L(p_{i+1},j) &= L(p_i, m+1) - L(p_{i+1}, m) \\ &\quad + \sum_{j=2}^{m} L(p_i,j) - \sum_{j=1}^{m-1} L(p_{i+1},j) \\ &\le \sum_{j=2}^{m} L(p_i,j) - \sum_{j=1}^{m-1} L(p_{i+1},j).\end{aligned} \tag{7}$$

Following this conclusion, we split Eq. (3) and get:

$$\begin{aligned}&\max_t \left(\sum_{j=2}^{t} L(p_i,j) - \sum_{j=1}^{t-1} L(p_{i+1},j), 0 \right) \\ &= \max \left\{ \max_n \left(\sum_{j=2}^{n} L(p_i,j) - \sum_{j=1}^{n-1} L(p_{i+1},j), 0 \right), \right. \\ &\qquad \left. \left(max \left(\sum_{j=2}^{m+1} L(p_i,j) - \sum_{j=1}^{m} L(p_{i+1},j), 0 \right) \right) \right\} \\ &= \max_n \left(\sum_{j=2}^{n} L(p_i,j) - \sum_{j=1}^{n-1} L(p_{i+1},j), 0 \right).\end{aligned} \tag{8}$$

Here $2 \le \mathrm{t} \le \mathrm{m}+1$, $2 \le \mathrm{n} \le \mathrm{m}$, p_i and p_{i+1} are adjacent packets in the given packet scheduling sequence. We get the conclusion from Eq. (3) and Eq. (8) that the addition of the $(\mathrm{m}+1)$-th resource does not influence the start time of each packet. In this setting, the given packet scheduling sequence possesses the minimal $\mathbb{E}$ even in a $(\mathrm{m}+1)$-resource environment. Actually in middleboxes, packets will be put into memory after being processed on the CPU, then, they will be transferred to the link. With highly sufficient link bandwidth, the packet processing time on the link will be the smallest among all the resources. Under this setting, our efficient scheduling algorithm is also the optimal in a three-resource environment, which is the common case in middleboxes.

For getting the minimal $\mathbb{E}$ in each scheduling loop, we adopt the algorithm [35] proposed by Gilmore and Gomory, which is originally proposed in the field of operation research. We first generalize it into the packet scheduling problem. Algorithm 1 describes the details of the scheduling algorithm.

Algorithm 1 Efficient Scheduling Algorithm

Require: $\mathcal{P} =< p_1, p_2, \cdots, p_{N+1} >$: the packet scheduling sequence; A_i: the packet processing time of p_i on the first resource; B_i: the packet processing time of p_i on the second resource; $\mathcal{C} = \{c_1, c_2, \cdots, c_N\}$: the cost array; $subsequenceN = 0$: the number of packet subsequences; $\mathcal{R}$: number set; $\mathcal{S}$ and $\mathcal{T}$: two number sequences.

1: Arrange and renumber the packets in $\mathcal{P}$ according to the value of B_i, so that in the new scheduling sequence, $B_i \le B_{i+1}$, and $i = 1, 2, \ldots, N$;
2: Arrange the value of A_i in increasing order;
3: **for** each $p_m \in \mathcal{P}$ **do**
4: $\quad \varphi(m) = n$;
5: $\quad$ **if** (new packet subsequence arises) **then**
6: $\quad\quad subsequenceN \leftarrow subsequenceN + 1$;
7: **for** each $c_i \in \mathcal{C}$ **do**
8: $\quad$ **if** $\max(B_i, A_{\varphi(i)}) \ge \min(B_{i+1}, A_{\varphi(i+1)})$ **then**
9: $\quad\quad c_i = 0$;
10: $\quad$ **else**
11: $\quad\quad c_i = \min(B_{i+1}, A_{\varphi(i+1)}) - \max(B_i, A_{\varphi(i)})$;
12: **while** $subsequenceN \ne 1$ **do**
13: $\quad$ Select the minimal c_i obeying that p_i belongs to one packet subsequence and p_{i+1} belongs to another;
14: $\quad$ Combine these two packet subsequences together;
15: $\quad$ Add c_i in $\mathcal{R}$;
16: $\quad subsequenceN \leftarrow subsequenceN - 1$;
17: **for** each $c_i \in \mathcal{R}$ **do**
18: $\quad$ **if** $A_{\varphi(i)} \ge B_i$ **then**
19: $\quad\quad$ Add i in $\mathcal{S}$;
20: $\quad$ **else**
21: $\quad\quad$ Add i in $\mathcal{T}$;
22: Arrange the numbers in $\mathcal{S}$ in decreasing order and get $\mathcal{S} =< s_1, s_2, \cdots >$;
23: Arrange the numbers in $\mathcal{T}$ in increasing order and get $\mathcal{T} =< t_1, t_2, \cdots >$;
24: **for** each $p_m \in \mathcal{P}$ **do**
25: $\quad \psi(m) = \mathcal{X}_{s_1,s_1+1}\mathcal{X}_{s_2,s_2+1}\cdots\mathcal{X}_{t_1,t_1+1}\mathcal{X}_{t_2,t_2+1}\cdots\varphi(m)$;

As for this algorithm, the packet scheduling sequence $\mathcal{P} = < p_1, p_2, \cdots, p_{N+1} >$ contains all of the N packets in the same scheduling loop and these packets can be arranged randomly at the beginning. Each $p_i \in \mathcal{P}$ associates with two values, A_i and B_i, which denote its processing time on the first and second resources, respectively. In addition, a virtual packet p_{N+1} will be added into $\mathcal{P}$. A_{N+1} and B_{N+1}will be set as:

$$\begin{aligned} A_{N+1} &= \max_{1 \le i \le N} B_i, \\ B_{N+1} &= \begin{cases} B_{N'} \\ 0 \end{cases} \end{aligned} \tag{9}$$

where $p_{N'}$ is the last packet scheduled in the previous scheduling loop and $B_{N'}$ is the processing time of $p_{N'}$ on the second resource. In the first scheduling loop, B_{N+1} will be set as 0. The introduced virtual packet only takes part in the computation, but will never be processed in actual.

In step 4, n is the subscript of the packet in $\mathcal{P}$ and A_n is the m-th smallest processing time on the first resource. By doing this, each $p_m \in \mathcal{P}$ gets an original successor, i.e., the next packet to be processed after it. $\varphi(m)$ indicates the subscript of the original successor of p_m. Consequently, some packet subsequences arise and they are independent with each other. The cost array $\mathcal{C}$ indicates the cost for combining two packet subsequences together and can be got from step 7 to step 11. At step 14, two packet subsequences are combined together by exchanging the successors of p_i and p_{i+1}. By picking out the minimal c_i in every iteration, we combine all the packet subsequences together with the minimal cost. After combining all the packet subsequences together, the original successors of packets need to be adjusted through the formula described at step 24, so as to get the final successors for them. Here the operation of $\mathcal{X}_{a,b}(z)$ is defined as:

$$\mathcal{X}_{a,b}(z) = \begin{cases} a & z = b \\ b & z = a \\ z & z \neq a \text{ or } b. \end{cases} \tag{10}$$

The subscript of p_m's successor can be got as $\psi(m)$ by applying $\mathcal{X}_{a,b}(z)$ from right to left to $\varphi(m)$. All the successors of packets in $\mathcal{P}$ can be got in the same way. The scheduling algorithm combines all the packet subsequences together, consequently we achieve the packet scheduling sequence with the minimal $\mathbb{E}$. The successor of the virtual packet will be processed first in the current scheduling loop and other packets will be processed sequentially.

C. Analysis About the Properties of ATFQ

ATFQ pursues high efficiency with the guarantee of the fairness. We analyze its emphasis on the fairness and the tradeoff between the fairness and the efficiency in Section IV-C1 and Section IV-C2, respectively.

1) ATFQ's Emphasis on the Fairness: The ATF and the time domain bring ATFQ the following attractive properties, which are essential for the packet scheduling methods as mentioned in Section III-A.

1) All flows obtain the same proportion of their anticipated rates. Assume that a given set of packets, belonging to different flows, obey the time domain $[T1, T2)$. The total packet size of flow f_i can be denoted by SP_i. If flow f_i monopolizes all the resources, its average data rate in the time domain can be expressed as:

$$r_i = \frac{SP_i}{T2 - T1}, \tag{11}$$

where r_i can be seen as the anticipated data rate for flow f_i. When flows share the multiple resources with each other, their anticipated data rate cannot be achieved concurrently. Let T denote the makespan for processing all the given packets under any scheduling method. Obviously, T is larger than $(T2 - T1)$ in most cases. Thus, the actual average rate of flow f_i can be expressed as:

$$r_i' = \frac{SP_i}{T}. \tag{12}$$

Compared with r_i, we get:

$$r_i' = \frac{T2 - T1}{T} * r_i. \tag{13}$$

This formula indicates that each flow actually gets the same proportion of its anticipated rate. It is fair because all flows are treated equally. What's more, although applying any scheduling methods to the given set of packets results in the same conclusion, our efficient scheduling algorithm pursues the minimal T to maximize this proportion.

2) Flows cannot get more service by cheating. In an untrusted network, illegal flows strive to get more service by requiring more resources. However, ATFQ computes the timestamps of packets belonging to different flows respectively. If a flow intentionally enlarges its processing time requirements on some resources, the timestamps of its packets will stay the same or be magnified, according to Eq. (3) and Eq. (5). Constrained by the size of the time domain, this flow will get the same or fewer opportunities to be processed in this scheduling loop.
3) No flow will be starved. The first packet of a newly arriving flow will be allocated the maximal timestamp as in Eq. (1), so as to be processed as soon as possible. Packets obeying the time domain will be processed in the current scheduling loop. If the size of the time domain is set suitably, each flow can get at least one opportunity of being processed in every scheduling loop.

2) Tradeoff Between the Fairness and Efficiency: As we discussed previously, for maximizing the efficiency, we introduce the concept of time domain to relax the constraint of the fairness. Actually, the degree of the tradeoff between the fairness and the efficiency is considerably influenced by the size of the

TABLE II
THE PROCESSING TIME ON THE CPU FOR DIFFERENT APPLICATIONS

Application	CPU Processing Time (μs)
IPSec Encryption	0.015x+84.5
Statistical monitoring	0.0008x+12.1
Basic forwarding	0.00286x+6.2
Redundancy Elimination	0.006987x+10.97

time domain. Assume two adjacent scheduling loops, denoted by SL_1 and SL_2. They are associated with the time domain $[T1, T2)$ and $[T2, T3)$, respectively. The efficient scheduling algorithm can get their respective minimal $\mathbb{E}$, denoted by $\mathbb{E}_1$ and $\mathbb{E}_2$. The packets in SL_1 and SL_2 will be scheduled together in a new scheduling loop, denoted as SL_3, if we adopt the time domain $[T1, T3)$. We can also get the minimal $\mathbb{E}$ for SL_3, denoted by $\mathbb{E}_3$. Obviously, the sum of $\mathbb{E}_1$ and $\mathbb{E}_2$ must be no less than $\mathbb{E}_3$. Thus, we get the conclusion that a long time domain prefers more to the efficiency.

On the contrary, a short time domain enables fewer packets to enter the scheduling loop. The efficient scheduling algorithm is agnostic to the fairness. So, processing fewer packets in one scheduling loop will shorten the waiting time between each flow's adjacent packets and the difference of flows' received service time. In an extreme case, processing just one packet in every scheduling loop realizes the strict fairness, just as what DRFQ does. In this setting, the efficient scheduling algorithm is disabled and ATFQ degrades to pursue only the fairness. So, we conclude that a short time domain prefers more to the fairness.

Based on the above discussions, the size of the time domain can be taken as a metric to measure the degree of the tradeoff between the fairness and the efficiency. This will be explicitly discussed in Section V.

V. PERFORMANCE EVALUATION

We first describe the experiment settings in details. We then evaluate the fairness of ATFQ, including the data rate allocation and the service difference. We finally evaluate the efficiency of ATFQ by measuring the packet delay, the packet completion time and the resource utilization.

A. Experiment Settings

For comparing ATFQ with DRFQ, we follow the measurement data used in DRFQ. Middleboxes perform a variety of functions. The packets undergoing different functions have variable processing time on diverse resources. However, the packet processing time on diverse resources follows an approximate linear relationship to the packet size [4]. On a specified resource, it can be denoted as $\alpha x + \beta$, where x is the size of the packet, α and β are parameters associating with the function. Table II lists the CPU processing time for packets undergoing different functions.

In our evaluation, packets will be sent to the link after being processed on the CPU. We set the link bandwidth as 400 Mbps, so as to congest flows on the CPU and the link differently. What's more, we process x packets in every scheduling loop and denote this implementation of ATFQ as ATFQ-x. The reason

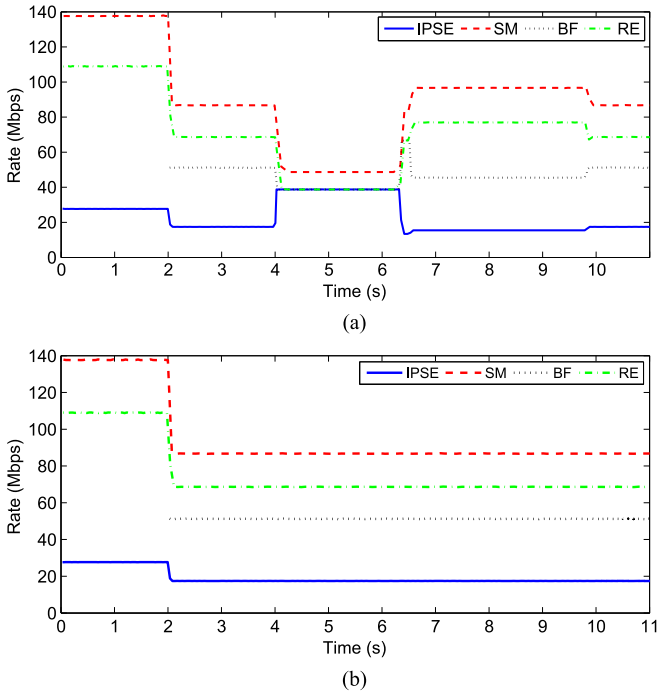


Fig. 5. Data rate allocation for flows. (a) Data rate allocation under the strict DRFQ. (b) Data rate allocation under the strict ATFQ.

for doing this is to avoid the packet reordering problem. For TCP flows, their packets should be transmitted and received sequentially. However, the efficient scheduling algorithm is powerless for this. Allowing at most one packet of each flow to enter the scheduling loop and correspondingly changing the value of x can perfectly solve this problem. It should be noted that ATFQ-x is not an off-line scheduling method. Given an appropriate value of x, there will be enough packets in the buffer space. All the packets in the queue will be processed in one scheduling loop if the number of packets is less than x.

B. Data Rate Allocation for Flows

We measure the achieved data rate for flows under the strict DRFQ and ATFQ-1. The strict DRFQ pursues the equalization of the processing time on flows' dominant resources and is vulnerable to the shift of the dominant resources. We simply denote the four applications listed in Table II as IPSE, SM, BF and RE. Four flows undergo each of these four applications and their packet sizes are set as 1000 bytes, 800 bytes, 200 bytes and 600 bytes, respectively. At the beginning, three flows resulting from IPSE, SM and RE arrive. After two seconds, the flow resulting from BF arrives. At the fourth second, each flow exchanges its processing time on the CPU and the link to shift its dominant resource.

As shown in Fig. 5(a), in the first two seconds, each of the three flows gets nearly 34.4% of its anticipated data rate under the strict DRFQ. From the second to the fourth second, each of the four flows gets nearly 21.6% of its anticipated data rate. Until now, DRFQ reasonably allocates the data rate for flows and each of them achieves almost the same proportion of its

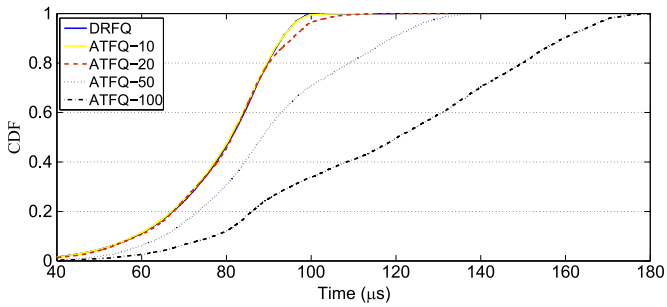


Fig. 6. The maximal difference of flows' received active time.

anticipated data rate. However, after the fourth second, the four flows encounter drastic jitter on their data rate for strictly pursuing the equivalent of the processing time on their respective dominant resources. At almost the tenth second, the data rate allocation recovers to the normal level. This phenomenon also verifies that DRFQ is vulnerable to the shift of the dominant resource. Fig. 5(b) plots the scheduling result under the strict ATFQ. At the beginning, each of the three flows also gets nearly 34.4% of its anticipated data rate. After two second, the forth flow arrives and each of the four flows gets nearly 21.6% of its anticipated data rate and keeps it until the end. In other words, ATFQ is immune to the shift of the dominant resource and provides more fair and stable data rate allocation for flows in a changeable scenario.

C. The Difference of Flows' Received Active Time

The difference of flows' received service time is the most important metric to measure the fairness. Here we take the active time as the service time. In the following experiments, we consider 20 flows, resulting from the four aforementioned applications, to compare the performance of DRFQ and ATFQ. Flows 1–5 belong to the IPSE. Their packet sizes are set as 200 bytes, 400 bytes, 600 bytes, 800 bytes and 1000 bytes, respectively, so as to bottleneck their packets on different resources. With the same setting of the packet size, flows 6–10, flows 11–15 and flows 16–20 belong to the SM, BF, and RE, respectively. Note that such 20 flows are backlogged and contain totally one million packets.

We measure the maximum and the minimum of flows' received active time and draw their difference in Fig. 6. The value of the maximal difference is updated and recorded when a packet completes its processing on all the resources.

Under the strict DRFQ and ATFQ-1, the maximal difference of flows' received active time can be bounded within a small range. Their distinction is so small that we only depict the strict DRFQ in Fig. 6. Even ATFQ-10 overlays with the strict DRFQ closely. In the case of ATFQ, the difference among flows' received active time directly depends on the number of packets processed in each scheduling loop. Under ATFQ-10 and ATFQ-20, this difference is very small, roughly below 100 μs. So they make almost the same fairness guarantee as the strict DRFQ. Processing more packets in each scheduling loop accordingly increases this difference.

D. The Delay Between Adjacent Packets

As for a flow, the delay between its adjacent packets, i.e., the interval between the start times of its adjacent packets, can be seen as a metric to evaluate its received scheduling opportunities. Some real-time applications and delay sensitive flows cannot tolerate a long delay between adjacent packets.

For the given 20 flows, we measure the delay between each flow's adjacent packets. Without loss of generality, we select four flows undergoing different functions, i.e., flow 3, flow 8, flow 13 and flow 18, and illustrate the measurement results in Fig. 7. Intuitively, ATFQ may have a longer delay between adjacent packets than DRFQ, since it relaxes the constraint of the fairness to improve the efficiency. However, benefiting from higher resource utilization, all flows achieve a shorter average delay between adjacent packets under ATFQ-20 than the strict DRFQ. Flow 3 has the maximal processing time difference on diverse resources and maximally shortens the delay between its adjacent packets. Obviously, with shorter inter-packet delay, all the flows can be transmitted quicker.

E. The Promotion of the Packet Completion Time

An efficient scheduling method can process more flows or packets within the same time interval. In other words, as for the same packet, it gets earlier completion time under a more efficient scheduling method. So we measure the completion time for the given one million packets to quantize the time saving under ATFQ, compared to DRFQ.

Actually, as our evaluations verified, all the 20 flows get the approximate promotion on the packet completion time. Here we take flow 1 as an example and illustrate the measurement results in Fig. 8. Under the strict DRFQ, the final completion time for processing 10 thousands of its packets is roughly 18.1 seconds. But under ATFQ-20, the final completion time for processing the same number of packets is 16.3 seconds, saving up almost 10% of the processing time than DRFQ. Meanwhile, under the strict DRFQ, about 9900 packets can be processed in 18 seconds, but AFTQ-20 processes 11000 packets within the same time. This is the most important benefit resulting from relaxing the fairness to improve the efficiency. It is worth noting that the total completion time of the packets in each scheduling loop is improved with approximate extent (roughly 10%) by using ATFQ-20. Compared to the total but short processing time of packets in the current scheduling loop, the promotion of the completion time of each packet can be measured as the sum of the time saving in the previous scheduling loops. Thus, the packet completion time under ATFQ has a linear improvement, compared to DRFQ.

F. The Promotion of the Resource Utilization

Finally, we measure the resource utilizations of the CPU and the link under DRFQ and ATFQ. As illustrated in Fig. 9, only 86.42% of the CPU and 61.84% of the link are utilized under the strict DRFQ. Meanwhile, ATFQ-1 achieves nearly the same resource utilizations as DRFQ. In the case of ATFQ-5, the resource utilizations of the CPU and the link respectively

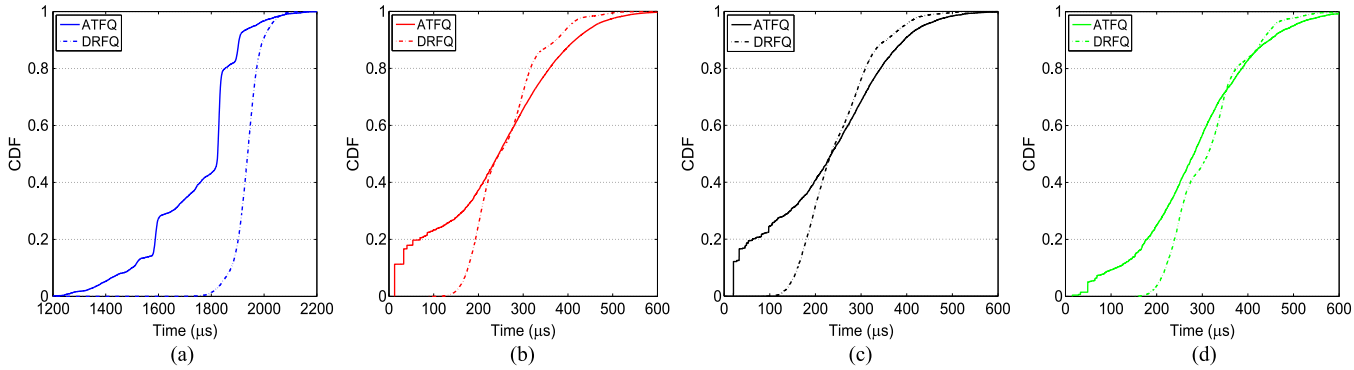


Fig. 7. The CDF of delay between adjacent packets. (a) Flow 3. (b) Flow 8. (c) Flow 13. (d) Flow 18.

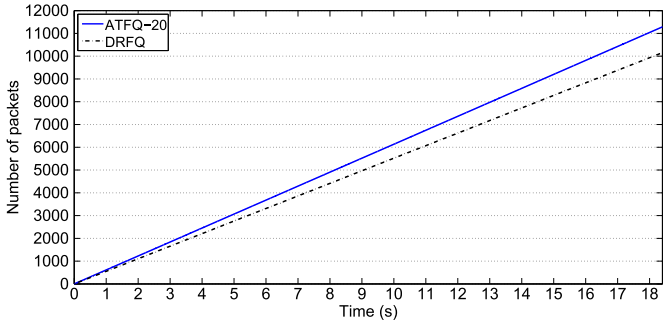


Fig. 8. Promotion of the completion time for packets of flow 1.

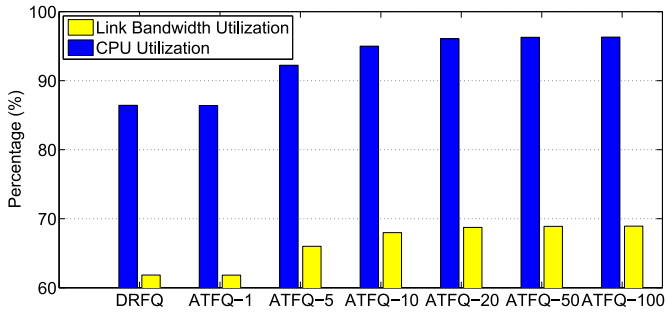


Fig. 9. The resource utilizations under DRFQ and ATFQ.

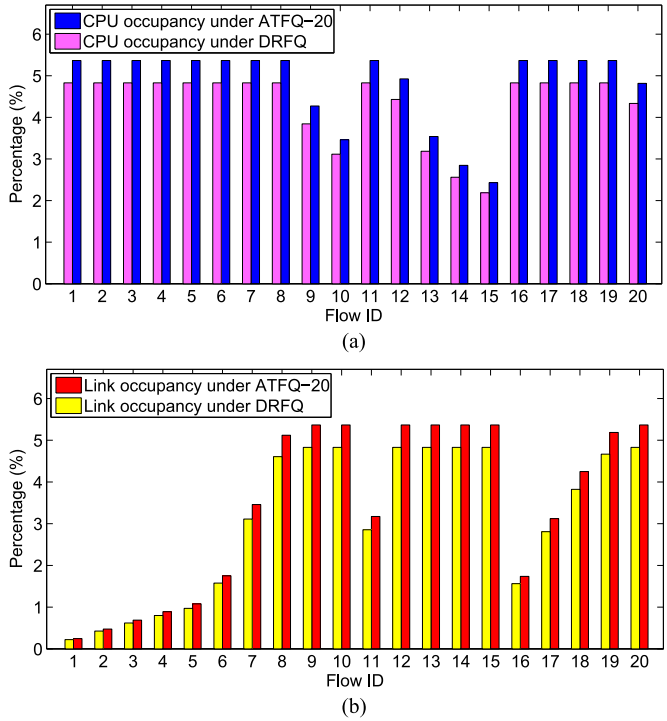


Fig. 10. The resource occupancy for flows under DRFQ and ATFQ. (a) The CPU occupancy for flows. (b) The link occupancy for flows.

grow up to 92.23% and 66%. Furthermore, under ATFQ-20, the resource utilizations rise up to 96.06% and 68.74%. That is, processing more packets in every scheduling loop will improve the resource utilization. However, there is an upper bound for the promotion of the resource utilization, because not all the time gap can be eliminated. Furthermore, we measure the resource occupancy, i.e., the ratio of the processing time to the final completion time, for the given 20 flows under the strict DRFQ and ATFQ-20. As illustrated in Fig. 10, since flows have shorter completion time under ATFQ than DRFQ, they get higher resource occupancy under ATFQ. What's more, ATFQ realizes more reasonable resource allocation than DRFQ. That is, those CPU-bounded flows get more promotion on the CPU occupancy and those link-bounded flows achieve more promotion on the link occupancy.

To sum up, although scheduling more packets in each scheduling loop can more improve the resource utilization, such a trend drops off with the growing value of x. With the given network load in our experiments, scheduling 10∼20 packets in each scheduling loop can sufficiently improve the resource utilization (roughly 10%). Meanwhile, the maximal difference of flows' received active time will be bounded within a small range (roughly below 100 μs), approximate to DRFQ. This is an ideal range to select the value of x.

VI. Conclusion

In the setting of a multi-resource environment, how to ensure the quality of service for flows comes out to be a challenge in recent years and attracts lots of attention. Prior packet scheduling methods take the fairness as the only scheduling objective, but suffer from enormous waste of resources, which on the contrary influences the quality of service for flows. In this paper, we discover that the fairness and the scheduling

efficiency are not opposite. With the guarantee of the former, the latter can be improved by a large extent. Based on this observation, we design ATFQ, a novel packet scheduling method in a multi-resource environment, and take both the fairness and the efficiency as the scheduling objectives. As verified in the performance evaluation, flows can achieve fair service under the guarantee of the active time fairness. What's more, benefiting from higher resource utilization, all the flows achieve better QoS in many aspects.

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, pp. 1–14.

[2] J. Sherry *et al.*, "Making middleboxes someone elses problem: Network processing as a cloud service," in *Proc. ACM SIGCOMM*, 2012, pp. 1–12.

[3] K. Argyraki *et al.*, "Understanding the packet forwarding capability of general-rurpose processors," Intel Res. Berkeley, CA, USA, Tech. Rep. IRB-TR-08-44, 2008.

[4] A. Ghodsi, V. Sekar, M. Zaharia, and I. Stoica, "Multi-resource fair queueing for packet processing," in *Proc. ACM SIGCOMM*, 2012, pp. 1–12.

[5] W. Wang, B. Liang, and B. Li, "Multi-resource generalized processor sharing for packet processing," in *Proc. ACM/IEEE IWQoS*, 2013, pp. 1–10.

[6] W. Wang, B. li, and B. Liang, "Multi-resource round robin: A low complexity packet scheduler with dominant resource fairness," in *Proc. ICNP*, 2013, pp. 1–10.

[7] S. K. Baruah, N. K. Cohen, C. G. Plaxton, and D. A. Varvel, "Proportionate progress: A notion of fairness in resource allocation," in *Algorithmica*, vol. 15, no. 6, pp. 600–625, Jun. 1996.

[8] A. Parekh and R. Gallager, "A generalized processor sharing approach to flow control: The single node case," *ACM Trans. Netw.*, vol. 1, no. 3, pp. 344–357, Jun. 1993.

[9] A. Demers, S. Keshav, and S. Shenker, "Analysis and simulation of a fair queueing algorithm," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 19, no. 4, pp. 1–12, Sep. 1989.

[10] J. Bennett and H. Zhang, "WF^2Q: worst-case fair weighted fair queueing," in *Proc. IEEE INFOCOM*, 1996, pp. 120–128.

[11] S. Golestani, "A self-clocked fair queueing scheme for broadband applications," in *Proc. IEEE INFOCOM*, 1996, pp. 636–646.

[12] P. Goyal, H. M. Vin, and H. Chen, "Start-time fair queueing: A scheduling algorithm for integrated services packet switching networks," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 26, no. 4, pp. 157–168, 1996.

[13] N. Dukkipati, M. Kobayashi, R. Zhang-Shen, and N. McKeown, "Processor sharing flows in the Internet," in *Proc. IWQoS*, 2005, pp. 271–285.

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

[15] N. Dukkipati, N. McKeown, and A. G. Fraser, "RCP-AC: Congestion control to make flows complete quickly in any environment," in *Proc. IEEE INFOCOM*, 2006, pp. 1–3.

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

[17] A. Munir *et al.*, "Minimizing flow completion times in data centers," in *Proc. IEEE INFOCOM*, 2013, pp. 2157–2165.

[18] C.-Y. Hong, M. Caesar, and P. B. Godfrey, "Finishing flows quickly with preemptive scheduling," in *Proc. ACM SIGCOMM*, 2012, pp. 127–138.

[19] D. Zats, T. Das, P. Mohan, D. Borthakur, and R. Katz, "Detail: Reducing the flow completion time tail in datacenter networks," in *Proc. ACM SIGCOMM*, 2012, pp. 139–150.

[20] M. Alizadeh *et al.*, "pfabric: Minimal near-optimal datacenter transport," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 43, no. 4, pp. 435–446, Oct. 2013.

[21] W. Bai *et al.*, "Information-agnostic flow scheduling for commodity data centers," in *Proc. USENIX NSDI*, 2015, pp. 1–14.

[22] A. Ford, C. Raiciu, M. Handley, and O. Bonaventure, "Tcp extensions for multipath operation with multiple addresses," Internet Eng. Task Force (IETF), Fremont, CA, USA, Internet-Draft, 2011.

[23] D. Wischik, C. Raiciu, A. Greenhalgh, and M. Handley, "Design, implementation and evaluation of congestion control for multipath TCP," in *Proc. NSDI*, 2011, pp. 1–14.

[24] C. Raiciu *et al.*, "Improving datacenter performance and robustness with multipath TCP," in *Proc. ACM SIGCOMM*, 2011, pp. 266–277.

[25] H. Xu and B. Li, "Repflow: Minimizing flow completion times with replicated flows in data centers," in *Proc. IEEE INFOCOM*, 2014, pp. 1–9.

[26] J. Guo *et al.*, "On efficient bandwidth allocation for traffic variability in datacenters," in *Proc. IEEE INFOCOM*, 2014, pp. 1572–1580.

[27] J. Guo, F. Liu, J. Lui, and H. Jin, "Fair network bandwidth allocation in iaas datacenters via a cooperative game approach," in *Proc. IEEE/ACM ToN*, 2015, p. 1.

[28] C. Wilson, H. Ballani, T. Karagiannis, and A. Rowstron, "Better never than late: meeting deadlines in datacenter networks," in *Proc. ACM SIGCOMM*, 2011, pp. 50–61.

[29] B. Vamanan, J. Hasan, and T. Vijaykumar, "Deadline-aware datacenter tcp (D^2TCP)," in *Proc. ACM SIGCOMM*, 2012, pp. 115–126.

[30] A. Ghodsi *et al.*, "Dominant resource fairness: Fair allocation of multiple resource types," in *Proc. IEEE NSDI*, 2011, pp. 1–14.

[31] W. Wang, B. Liang, and B. Li, "Low complexity multi-resource fair queueing with bounded delay," in *Proc. IEEE INFOCOM*, 2014, pp. 1914–1922.

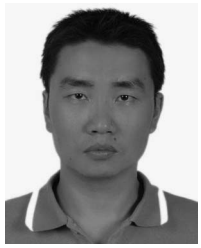
[32] X. Li and C. Qian, "Low-complexity multi-resource packet scheduling for network function virtualization," in *Proc. IEEE INFOCOM*, 2015, pp. 1–9.

[33] C. Joe-Wang, S. Sen, T. Lan, and M. Chiang, "Multi-resource allocation: Fairness-efficiency tradeoffs in a unifying framework," in *Proc. IEEE INFOCOM*, 2012, pp. 1206–1214.

[34] L. Zhang, "Virtual clock: A new traffic control algorithm for packet switched networks," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 20, no. 4, pp. 19–29, Sep. 1990.

[35] P. C. Gilmore and R. E. Gomory, "Sequencing a one state-variable machine: A solvable case of the traveling salesman problem," in *Oper. Res.*, vol. 12, pp. 655–679, 1964.

Jianhui Zhang received the B.E. degree from the School of Computer Science and Technology, Harbin Engineering University, Harbin, China, in 2009. Currently, he is pursuing the Ph.D. degree in the School of Computer Science and Technology, Dalian University of Technology, Dalian, China. His research interests include datacenter networks, network protocols and cloud computing.



Heng Qi received the bachelor's degree from Hunan University, Hunan, China, in 2004 and the master's and doctoral degrees from Dalian University of Technology, Dalian, China, in 2006 and 2012, respectively. He was a Lecturer at the School of Computer Science and Technology, Dalian University of Technology. He served as a software engineer in GlobalLogic-3CIS from 2006 to 2008. His research interests include computer network, multimedia computing, and mobile cloud computing. He has published more than 20 technical papers in international journals and conferences.

Deke Guo received the B.S. degree in industry engineering from Beijing University of Aeronautic and Astronautic, Beijing, China, in 2001, and the Ph.D. degree in management science and engineering from National University of Defense Technology, Changsha, China, in 2008. He is an Associate Professor with the College of Information System and Management, National University of Defense Technology, Changsha, China. His research interests include distributed systems, software-defined networking, data center networking, wireless and mobile systems, and interconnection networks.

Keqiu Li (SM'14) received the bachelor's and master's degrees from the Department of Applied Mathematics at the Dalian University of Technology, Dalian, China, in 1994 and 1997, respectively. He received the Ph.D. degree from the Graduate School of Information Science, Japan Advanced Institute of Science and Technology, Nomi, Japan, in 2005. He also has two-year postdoctoral experience with the University of Tokyo, Japan. He is currently a Professor in the School of Computer Science and Technology, Dalian University of Technology. He has published more than 100 technical papers, such as IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, *ACM Transactions on Internet Technology*, and *ACM Transactions on Multimedia Computing, Communications, and Applications*. He is an Associate Editor of IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS and IEEE TRANSACTIONS ON COMPUTERS. His research interests include Internet technology, data center networks, cloud computing, and wireless networks.

Wenxin Li received the B.E. degree from the School of Computer Science and Technology, Dalian University of Technology, Dalian, China, in 2012. Currently, he is pursuing the Ph.D. degree in the School of Computer Science and Technology, Dalian University of Technology. His research interests include datacenter networks and cloud computing.

Yingwei Jin received the doctoral degree from Dalian University of Science and Technology, Dalian, China, in 2005.

He is currently a Professor with the School of Management, Dalian University of Technology. His research interests include computer network and security, Internet technology, and artificial intelligence. He has authored or coauthored over 30 technical papers in international journals and conferences.