MDFS: Deadline-Driven Flow Scheduling Scheme in Multi-Resource Environments

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

Abstract—Data centers have emerged as infrastructures for deploying various applications and services. To improve the security and performance, middleboxes are vastly deployed inside data centers to perform a large range of functionalities. Each of such middleboxes is equipped with diverse resources. Compared with traditional switches, middleboxes analyze the content of packets. This leads to the long processing time for flows passing through a middlebox. Additionally, executing different functionalities incurs diverse consumption on resources. Consequently, data flows undergoing different function components need different processing time on diverse resources. How to complete the transmission of such flows before their deadlines when passing through a middlebox comes out to be an essential issue, which lacks effective solutions. In this paper, we propose multi-resource & deadline-driven flow scheduling (MDFS) to satisfy the deadline requirements of flows in multi-resource environments. Besides guaranteeing the deadline, MDFS treats flows fairly and provides reliable service for them. To the best of our knowledge, this is the first paper trying to solve the deadline-driven flow scheduling problem in a multi-resource environment. With respect to the performance evaluation, up to 90 percent flows meet their deadlines in normal conditions by using MDFS, which greatly outperforms the performance of other scheduling schemes.

Index Terms—Deadline-driven, fairness, scheduling algorithm, middlebox

1 Introduction

Large-scale data centers have served as the infrastructure for online applications and services, e.g., MapReduce, social network, and web search. Such applications and services usually have diverse requirements on the quality of service the data center provides. For example, online video service needs stable link bandwidth, web search has a strict restriction on the response delay, and applications for synchronizing the backup data need high link bandwidth. Ignoring such specific requirements of different applications not only influences the user experience, but also decreases the operator's revenue.

Meanwhile, middleboxes, which perform a large range of functionalities [1], [2], e.g., firewall, intrusion detection, and network address translation, are vastly deployed in data centers as a basic component. Different from traditional L2/L3 network devices, which just simply forward packets, they analyze the content of packets and then make corresponding processing. This leads to the long processing time for flows passing through a middlebox. What's more, middleboxes consist of multiple resources, e.g., CPU, memory, and link bandwidth. Executing different functionalities leads to diverse consumption on these resources. Consequently, flows undergoing different function components inside a middlebox result in different processing time on diverse resources. Additionally, flows stemming from different applications usually possess diverse requirements on quality of service, e.g., link bandwidth, delay, and deadline. As for flows deriving from soft-real applications, their transmission, including the waiting and processing time in a middlebox, should be completed before their deadlines. Otherwise, they will become invalid and be dropped. All these settings extremely complicate the flow scheduling problem in multi-resource environments. When flows undergo different function components of a middlebox, how to guarantee their requirements on some specific metrics comes out to be a challenge.

As for the flow scheduling problem in a single-resource environment, e.g., the link bandwidth, or the buffer queue, scheduling rules are made based on specified scheduling goals. The following three goals are usually considered. (1) Fairness. In different scheduling scenarios, the fairness can be defined in various modes, e.g., the equal allocation, the max-min fairness, and the proportional fairness. All flows should be treated fairly and receive fair service, according to any given fairness. (2) Average flow completion time. The completion time of a flow indicates the time interval from the beginning of its transmission to its receival by the target node. Given a set of flows, shorter average completion time indicates higher resource utilization and more flows can be completed within the same time interval. Some scheduling schemes also focus on alleviating the long-tailed distribution of the flow completion time. (3) Deadline. Soft real-time services have strict restriction on the completion time of their

• *J. Zhang, K. Li, H. Qi, and W. Li are with the School of Computer Science and Technology, Dalian University of Technology, No 2, Linggong Road, Dalian 116023, China. E-mail: {zhangjh, liwenxin}@mail.dlut.edu.cn, likeqiu@gmail.com, hengqi@dlut.edu.cn.*
• *D. Guo is with the School of Information System and Management, National University of Defense Technology, Changsha 410073, China. E-mail: dekeguo@nudt.edu.cn.*
• *Y. Jin is with the School of Management, Dalian University of Technology, Dalian 116023, China. E-mail: jinyw67@dlut.edu.cn.*

Manuscript received 23 June 2015; revised 8 Sept. 2015; accepted 19 Nov. 2015. Date of publication 7 Dec. 2015; date of current version 27 Jan. 2016. Recommended for acceptance by S. Pasricha.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TMSCS.2015.2506167

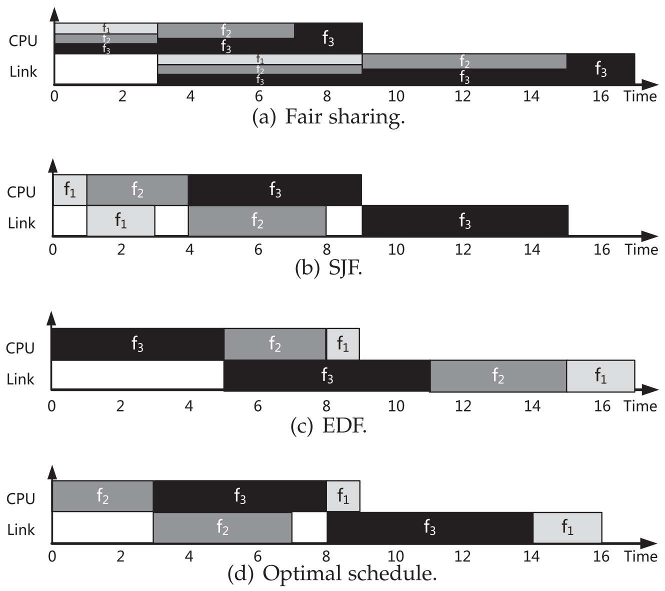


Fig. 1. Flow scheduling schemes in a multi-resource environment.

TABLE 1
Flow Information

FlowID	Arrive Time	CPU Time	Link Time	Deadline
f_1	0	1	2	16
f_2	0	3	4	15
f_3	0	5	6	14

tasks. As for web search, user's requests should be responded within an acceptable time interval. Otherwise, flows missing their deadlines will not be responded to the users. Consequently, this will severely influence the user experience. Moreover, some flow scheduling schemes also focus on the utilization of the link bandwidth and the load balance of the network.

In practice, data center operators always strive to guarantee the quality of service for applications by providing fair services. This scheme works well in some scenarios, but it severely restricts applications, which impose strict constraints on the deadline. Actually, flows with diverse deadlines should be treated differently, rather than a fair manner. Similarly, the scheduling schemes focusing on the average flow completion time are also unaware of the flow's deadline. Although the flows in the same set can be completed more quickly, some of them still miss their deadlines. Consequently, such two kinds of scheduling schemes cannot ensure the deadline requirements of flows in a single-resource environment, not to mention the multi-resource environments.

To ease the presentation, we use Fig. 1 as an illustrative example, where the setting of flows is listed in Table 1. For simplicity, each flow contains just one packet. Packets will be inputted into the link after being processed on CPU. In this setting, we get the following conclusions:

1) Fairness-driven schemes cannot satisfy the deadline requirements of flows in a multi-resource environment. Such schemes are unaware of the flow's deadline. They usually allocate equal processing time for flows, so as to provide fair service for them. As illustrated in Fig. 1a, we assume that resources can be multiplexed by flows simultaneously and hence implement fair sharing on all the resources. The three flows are completed at 9, 15 and 17, respectively. It is obvious that f_3 misses its deadline. That is, even under an ideal assumption, fair sharing cannot ensure the flows' deadlines in a multi-resource environment.
2) Those scheduling schemes, focusing on the average flow completion time, cannot satisfy the deadline requirements of flows in a multi-resource environment. Shortest job first (SJF) is agnostic to the flow's deadline and schedules flows according to the length of processing time on resources. In a single-resource environment, SJF achieves the minimal average flow completion time. Here we take the sum of the flow's processing time on all the resources as the metric and implement SJF in a multi-resource environment. As illustrated in Fig. 1b, the three flows are completed at 3, 8 and 15, respectively. This scheduling sequence achieves the minimal average flow completion time, but f_3 still misses its deadline. That is, not all the flows can meet their deadlines even though they obtain the minimal average flow completion time in a multi-resource environment.
3) Deadline-driven scheduling schemes in a single-resource environment are not suitable in multi-resource environments. Earliest deadline first (EDF) and least laxity first (LLF) are the optimal scheduling schemes for maximizing the number of flows, which meet their deadlines, in a single-resource environment. They prefer to process those flows with urgent deadlines and delay those flows with loose deadlines. However, such scheduling schemes are not suitable in multi-resource environments. As illustrated in Fig. 1c, the three flows are completed at 17, 15 and 11, respectively. However, f_1 misses its deadline. Consequently, we cannot adopt EDF or LLF to solve the deadline-driven flow scheduling problem in multi-resource environments.

For the optimal scheduling scheme, flows will be scheduled in the sequence illustrated in Fig. 1d, where all flows meet their deadlines. Many factors make the design of deadline-driven scheduling schemes very hard, especially in multi-resource environments. For example, flows need to be processed on multiple resources and incur diverse processing time on each resource. Some flows may need more time to be processed on the CPU, while others may consume more transmission time on the links. Meanwhile, resources in a middlebox are very limited. Flows usually contend for these resources drastically. This will be deteriorated when burst traffic emerges. In the worst case, only a few flows can meet their deadlines. After middleboxes are deployed in the network, the flow scheduling in a multi-resource environment becomes an essential challenge and such middleboxes will become the bottleneck. Consequently, middleboxes should be responsible to guarantee the deadline requirements of flows as many as possible.

However, scheduling schemes, which focus on the fairness or the average flow completion time, cannot guarantee the deadline requirements of flows in a multi-resource environment. Existing deadline-driven flow scheduling schemes focus on the setting of a single resource environment and bring enormous modifications on the senders, switches and receivers, e.g., D^3 [3]. Inspired by these observations, we design MDFS (multi-resource & deadline-driven flow scheduling) to solve the deadline-driven flow scheduling problem in multi-resource environments.

The major contributions of this paper are summarized as follows. We propose and characterize the deadline-driven flow scheduling problem in a multi-resource environment. For solving this problem, we propose MDFS, a deadline-driven flow scheduling scheme in a multi-resource environment. Besides guaranteeing the deadline requirements of flows, MDFS treats flows fairly and provides reliable service for them. It estimates the load of the middlebox to ensure that no flows miss their deadlines as long as they start to be processed. Finally, we verify the performance of MDFS through extensive simulations.

The rest of this paper is organized as follows. We introduce some traditional flow scheduling schemes in single-resource and multi-resource environment in Section 2. Then, we analyze the deadline-driven flow scheduling problem in multi-resource environment and the scheduling motivations of MDFS in Section 3. Section 4 explains the scheduling principles of MDFS in detail. Section 5 testifies the performance of MDFS on the availability, the reliability, and the fairness. Finally, we conclude our work in Section 6.

2 Related Work

We first introduce some traditional flow scheduling schemes. Then, we list some recently proposed flow scheduling schemes in multi-resource environments.

2.1 Traditional Flow Scheduling Schemes

Traditional flow scheduling schemes focus on the scheduling problem, where only a single type of resource is utilized to serve flows. According to the scheduling objectives, such scheduling schemes can be roughly categorized into two categories: the fairness-driven scheduling schemes and the time-driven scheduling schemes.

As for the former, the fairness can be defined variously, according to the corresponding scheduling scenarios and objectives. For example, the resource can be equally partitioned to all flows. However, max-min fairness prefers to guarantee the benefits of the flow with the minimal demand. Based on the specified definition of the fairness, the scheduler executes a fair partition on the resource or the processing time for flows.

Generally, the arrival of flows cannot be anticipated in advance. Thus, first come first serve (FCFS) can be seen as fair for the data flows in the simplest scheduling scenario. Fair queueing (FQ) [4] equally allocates the buffer queue in the switch for the passing flows to provide fair service. Obviously, this simple scheme cannot ensure the quality of service for flows when they have different priorities. Weighted fair queueing (WFQ) [5] partitions the buffer queue according to the weights of flows. Thus, the flow with weight w_i will get $\frac{w_i}{\sum_{j=1}^{n} w_j} * c$ of the buffer queue, where c is the capacity of the queue and n is the number of backlogged flows. Subsequently, start-time fair queueing (SFQ) [6], self-clocked fair queueing (SCFQ) [7] and worst-case fair weighted fair queueing (WF^2Q) [8] are proposed to improve the performance of the fair queueing algorithm under some specified scenarios.

Normally, only one packet can be processed on one resource at a time, e.g., the buffer queue. Thus, there will be a difference between flows' received service all the time. By virtue of a fluid model, generalized processor sharing (GPS) [9] assumes that all flows can be processed simultaneously on one resource. In this way, the buffer queue can be equally shared by all the backlogged flows. But GPS is too idealized to be implemented in practice and is sub-optimal to guarantee the end-to-end delay, compared with EDF [10].

As for the latter, it works when flows have strict requirements on some specified time factors, e.g., the flow's deadline, or the flow's completion time. Flows deriving from diverse applications possess different degrees of emergency. Thus, they should be scheduled differently, according to their deadlines. If a flow cannot be completed before its deadline, it will become invalid and be discarded. What's more, the resource captured by it will be wasted, which conversely slows down the completion of other flows. Existing deadline-driven flow scheduling schemes usually bring enormous modifications on senders, switches and receivers. These modifications limit the deployment of such schemes in data centers.

Literatures [11], [12], [13] focus on minimizing the flow's completion time by using rate control protocol (RCP). RCP implements the processor-sharing, i.e., allocating the same rate for all flows passing through the switch, on all switches in the network. D^3 [3] is a deadline-aware scheduling scheme and strives to guarantee the flows' deadlines through a fine-grained rate control. In its scheduling cycle, the scheduler continually adjusts the sending rate of the flow, according to the feedback information from the receiver. Firstly, each sender requires an expected sending rate, according to its remaining traffic and the deadline information, from switches on the path to the destination. Each switch allocates a feasible sending rate to such a sender. Based on the feedback information, the sender adjusts its sending rate, so as to complete the transmission before the flow's deadline.

Inspired by SJF and EDF, PDQ [14] schedules flows in a preemptive manner, according to their degrees of emergency. It minimize the flow's completion time on average and the number of flows missing their deadlines. It is implemented in a distributed manner, so as to reduce the overhead resulting from the centralized control. DeTail [15] focuses on the long tail distribution of the flow's completion time and alleviates this situation by prioritizing the latency-sensitive flows. As a variant of the traditional TCP, D^2TCP [16] is aware of the flows' deadlines. It leverages a novel congestion avoidance algorithm to modulate the size of the congestion window. As for the flows with loose deadlines, it will shrink vastly. Thus, D^2TCP also prefers the flows with more urgent deadlines.

Besides the size of data flows, unbalanced workload also influences the completion of flows. Transmitting flows along the congested links extremely delays the completion of flows. Consequently, all the idle links should be utilized in a reasonable manner. Multipath TCP (MPTCP) is a promising method. By adopting MPTCP [17], [18] can tackle the negative influence of unbalanced workload and flows can finish their transmission quickly.

2.2 Flow Scheduling Schemes in Multi-Resource Environments

The flow scheduling problem has not been explored very much in multi-resource environments. In this scenario, many factors, e.g., multiple resources and diverse requirements of flows, should be taken into account when making scheduling decisions. Such a setting extremely complicates the flow scheduling problem.

Generally, flow scheduling schemes get insights from the resource allocation problem. When different users have diverse requirements on resources, how to fairly partition multiple resources for users attracts lots of researchers' attention. Partition all the resource for users equally is the simplest manner. However, it ignores diverse requirements of users on different resources and leads to enormous waste of resources. An alternate method is to equally partition the bottleneck resource, i.e., the resource being mostly required. Other resources will be partitioned proportionally to the share of the bottleneck resource. However, this scheme is still coarse-grained and leads to waste of resources.

Recently, dominant resource fairness (DRF) [19] has been proposed to partition the multiple resources for users with diverse requirements on different resources. The dominant resource of a user indicates the resource with the maximal share among all the resources. It's worth noting that DRF possesses some attractive properties. It incentives users to share resources with others to get more service than equally partition of resources among users. In addition, envy-freeness makes each user never envy the allocated resources of other users. Otherwise, the received service of each user will be reduced.

Motivated by the attractive properties of DRF, some scheduling schemes, e.g., DRFQ [20], DRGPS [21], MR^3 [22], and GMR^3 [23], have been proposed to schedule flows in multi-resource environments. Although these scheduling schemes all take DRF as the definition of the fairness and focus on providing fair service for flows, they have distinct scheduling objectives. In detail, DRFQ and DRGPS generalize the concept of DRF into the flow scheduling problem in multi-resource environment and focus only on providing fair service isolation for flows passing through middleboxes. As typical timestamp based scheduling schemes, they need complex computations and comparisons, which lead to enormous scheduling overhead. Taking the implementation complexity into consideration, MR^3 integrates the round robin algorithm with DRF and makes scheduling decisions in O(1). However, MR^3 suffers the problem that a flow with higher priority has longer interval between its two packets. Later GMR^3 emerged to improve the weakness of MR^3 through a grouping algorithm. It decentralizes scheduling opportunities for flows according to their priorities.

Nonetheless, these schemes are all fairness-driven and focus only on providing fair service isolation for flows in essence. As we discussed previously, scheduling schemes

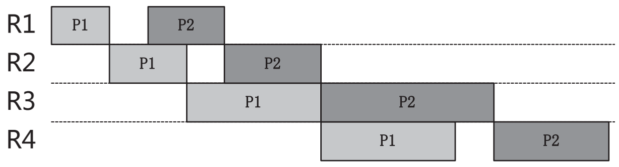


Fig. 2. The packet processing model.

focusing on fairness are agnostic to flows' deadline. So, they are not sufficient to ensure the deadline requirements for flows in multi-resource environments. Similarly extending the traditional scheduling schemes into multi-resource environments does not work when flows have diverse requirements on different resources. In this paper, we propose MDFS to solve the deadline-driven flow scheduling problem in multi-resource environments. It is aware of the flows' deadlines and makes fine-grained control over flows when making scheduling decisions. It's worth noting that MDFS provides reliable service for flows by using a precise prediction scheme. That is, only the flows with the possibility of being completed before their deadlines will be scheduled. In addition, it works in a non-preemptive manner, so as to avoid unnecessary waste of resources.

3 The Deadline-Driven Flow Scheduling in Multi-Resource Environments

We start with modeling the deadline-driven flow scheduling problem in a multi-resource environment. We then analyze the reasons why flows miss their deadlines. Along such analyses, we present the design motivations of MDFS.

3.1 Problem Statement

We first need to precisely define the packet processing procedure in multi-resource environments if we want to efficiently utilize the flow's deadline information. In this paper, we adopt the packet processing model illustrated in Fig. 2. We assume that all packets have to be processed successively on four kinds of resources, respectively denoted by $R1$, $R2$, $R3$ and $R4$. What's more, the packet processing procedure obeys the following principles:

1) Resources are not divisible. At any time, any resource can only process just one packet.
2) The processing of packets on any resource cannot be interrupted before their completion.
3) The start time of each packet on one resource, except for the first resource, is equal to its finish time on the previous resource.
4) All packets are processed along the same sequence on all the resources.
5) For avoiding unnecessary waste of resources, packets should be processed as early as possible on the first resource.

Actually, MDFS is not limited by the number of resources and the model can be extended. Although some aforementioned principles are not suitable for the middleboxes equipped with multi-core CPU, this concern exceeds the scope of this paper.

We formalize the flow scheduling problem in a multi-resource environment as follows. Let $F = \{f_1, f_2, \ldots, f_m\}$

denote the set of all backlogged flows, including the flows arriving within the time interval $[T1, T2)$ and the flows that have not been completed before $T1$. Here $T1$ and $T2$ are two time points, and $T1 < T2$. Logically, each flow can be denoted by a packet vector, i.e., $f_i = <p_i^1, p_i^2, \ldots, p_i^{z_i}>$. Here $p_i^{z_i}$ indicates the z_ith and also the last packet of f_i. Given any flow $f_i \in F$ and $1 \leq i \leq m$, a_i, d_i and s_i are three related parameters. a_i denotes the flow arriving time. Obviously, the values of a_i for all the newly arrived flows subject to the restriction that $T1 \leq a_i < T2$. d_i denotes the flow's deadline. s_i is a status denotation that indicates whether or not f_i will meet its deadline. If f_i cannot be completely processed before d_i, s_i will be set as 0. Otherwise, it will be set as 1.

Consider that data flows consist of packets. The flow scheduling problem can be converted as the problem of packet scheduling. As for every available packet scheduling sequence, denoted by AS_x, it contains all or part of the packets belonging to flows in set F, depending on the specified scheduling scheme. As for each flow, its packets in AS_x should obey the sequence in its packet vector. Correspondingly, each AS_x gets a value of:

$$g(AS_x) = \sum_{i=1}^{m} s_i. \tag{1}$$

It indicates the number of flows meeting their deadlines. The objective of a deadline-driven scheduling scheme is to maximize $g(AS_x)$ within the time interval $[T1, T2)$, i.e., getting the AS_* as:

$$\begin{aligned} AS_* = arg\ max\ g(AS_x) \\ s.t.\ T1 \leq t < T2. \end{aligned} \tag{2}$$

Actually, the design of a scheduling scheme should not only concentrate on maximizing the number of flows, which satisfy their deadlines. This will lead to other problems. Data flows usually differ in the size and deadline. In this setting, the flows with small sizes and urgent deadlines will preempt almost all the resources and other flows will be starved. So, the fairness and preemption should also be taken into consideration for a deadline-driven scheduler.

With respect to the fairness, the scheduler should treat flows in a fair manner and allocate scheduling opportunities equally among them. That is, the deadline of each flow cannot indicate its importance. Flows with urgent deadlines should not get more scheduling opportunities than flows with loose deadlines. The latter will be starved if the scheduler always prefers the former. When it comes to preemption, flows with urgent deadlines will preempt the resource occupied by other flows. This may make the latter miss their deadlines. In the worst case, all flows will miss their deadlines if this action repeats. So in our design, MDFS schedules flows in a fair and non-preemptive manner.

3.2 Causes of Missing Deadline

Taking the complex network environment into account, it is non-trivial to ensure the deadline requirements for all flow. Actually, flows may miss their deadlines due to the following reasons.

1) Deadline-agnostic. A lot of scheduling schemes have been proposed to enable fair service isolation for flows in the network. These scheduling schemes are agnostic to flow's requirements on deadline. TCP strives to provide fair service for all flows and achieves high utilization of the network. Nonetheless, fair service isolation cannot ensure the deadline requirements of flows. The deadline of a flow indicates its urgency. Thus, flows with different deadlines should also be treated differently. Similarly, those scheduling schemes focusing on the average flow completion time are also agnostic to a flow's deadline. Even if a set of flows can be processed within the minimal average flow completion time, some flows still miss their deadlines. Thus, deadline-agnostic scheduling schemes extremely hurt the benefits of flows with strict requirements on deadline.

2) The contention among flows on resources. When flows with requirements of deadline strive to be completed as soon as possible, the contention among flows on the limited resources will be intensified. Here resources can be the link bandwidth, the buffer queue or even the whole network. Constrained by the limited processing capability, the processing of flows preferred by the scheduler inevitably slows down the completion of other flows. If flows are scheduled in a reasonable sequence, their deadline requirements can be satisfied as many as possible. On the contrary, if the flows with loose deadlines preempt resources from the flows with urgent deadlines, as illustrated in Fig. 1b, the latter may miss their deadlines.

3) Burst traffic. Even for the optimal deadline-driven flow scheduling scheme, it cannot satisfy all flows' requirements of deadline under any scenario, e.g., setting the deadlines of all flows as 10 in Fig. 1. In other words, if the incoming traffic exceeds the processing capability of the middlebox, some flows will inevitably miss their deadlines. Data centers host all kinds of applications and services. Thus, the burst traffic is common in data centers. The contention among flows on resources will become acute when the burst traffic emerges. Under this setting, newly arrived flows may not be scheduled and finally miss their deadlines. Even worse, the flows, which have already been scheduled, may miss their deadlines, due to the influence of the newly arrived flows.

3.3 Design Motivations of MDFS

Based on the aforementioned discussions in Section 3.2, MDFS obeys the following three rationales:

1) Be aware of flow's deadline: The scheduler should be aware of flow's deadline. The deadline information should be taken into account when the scheduler makes scheduling decisions, so as to realize a fine-grained transport control for flows. In other words, flows with varied deadlines should be treated differently. As in D^3 [3], the sending scheme is determined based on flow's deadline and remaining size. The sender requires an expected rate from switches along the path to the destination endpoint as:

$$r_{t+1} = \frac{remaining\_flow\_length - s_t * rtt}{deadline - 2 * rtt}, \tag{3}$$

where s_t denotes the sending rate of a flow in the prior period and rtt denotes the round trip time. The sending rate

TABLE 2
Notations of Necessary Variables

Notation	Explanation
a_i	the arriving time of flow f_i
d_i	the deadline of flow f_i
s_i	the situation of flow f_i
z_i	the number of packet from flow f_i
p_i^k	the kth packet of flow i
$s(p_i^k)$	the size of packet p_i^k
$s(f_i)$	the size 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

in the next period will be justified according to the feedback information, resulting from the switches on the path. D^{2}TCP [16] computes a parameter d as:

$$d = \frac{T_c}{D}, \tag{4}$$

where T_c denotes the time to complete its transmission and D denotes the remaining time before its deadline. When the network meets traffic congestion, flows with larger d will be back-off at a small degree in the congestion avoiding period, and vice versa. So flows with urgent deadlines are preferred by D^{2}TCP. In summary, flows with diverse deadlines should be treated differently. The scheduler should be aware of the flow's deadline and utilize it to make scheduling decisions, so as to ensure that more flows can complete their transmission successfully before their deadlines.

2) Alleviating the contention among flows: In multi-resource environments, flows share the multiple resources with each other. Inevitably, the transmission of one flow will delay that of another flow. The contention may become acute especially when all flows strive to be complete their transmission quickly. Actually, the primary concern of flows is to be completed before their deadlines, rather than as soon as possible. Taking the web search as an example. Query results will be added into the final response as long as flows meet their deadlines. Shorter completion time of flows on average does not make the final response more precise. Alleviating the contention among flows can be realized by stretching the processing time of flows to their deadlines. If so, flows will contend for resources in a moderate manner. Under this setting, flows with loose deadlines will not preempt resources from flows with urgent deadlines. For stretching the processing time of flows, we have to know their original processing time.

Before further discussion, we summarize major notations and symbols in Table 2. Assume that there are n kinds of resources and they are numbered in the order of packet processing, e.g., packets will be pushed to the link after being processed on the CPU. In the simplest scenario, only one flow f_i passes through a middlebox and it exclusively occupies all resources. Meanwhile, f_i is backlogged. That is, p_i^{k+1} will arrive before p_i^k finishes its processing on the first resource. Based on such assumptions and the properties of the model we define in Section 3.1, p_i^1 will be processed when it arrives. Its start time on the first resource is given by

$$S(p_i^1, 1) = a_i. \tag{5}$$

The start time of other packets of f_i on the first resource is given by

$$\begin{aligned} S(p_i^{k+1}, 1) = {} & S(p_i^k, 1) + L(p_i^k, 1) \\ & + \max_r \left(\sum_{j=2}^{r} L(p_i^k, j) - \sum_{j=1}^{r-1} L(p_i^{k+1}, j), 0 \right), \end{aligned} \tag{6}$$

where $2 \leq r \leq n$. On the other resources, the start time of p_i^{k+1} is given by

$$S(p_i^{k+1}, r) = F(p_i^{k+1}, r-1). \tag{7}$$

The completion time of flow f_i on the last resource is given by

$$F(p_i^{z_i}, n) = S(p_i^{z_i}, 1) + \sum_{j=1}^{n} L(p_i^{z_i}, j). \tag{8}$$

For stretching the flow processing time to its deadline, we set:

$$F(p_i^{z_i}, n) = d_i. \tag{9}$$

Then we get $S(p_i^{z_i}, 1)$ as

$$S(p_i^{z_i}, 1) = d_i - \sum_{j=1}^{n} L(p_i^{z_i}, j). \tag{10}$$

That is, we delay the start time of $p_i^{z_i}$ on the first resource. The start time of other packets should be delayed correspondingly, so as to alleviate its aggressiveness on resources. This will be described in Section 4.1 in detail.

3) Be tolerant to the burst traffic: Due to the limited resources in middleboxes, the deadline requirements of flows cannot be always satisfied, especially when burst traffic emerges. The scheduler should be tolerant to the burst traffic in some degree. Actually, two strategies can be adopted to alleviate this influence:

(1) Accepting newly arrived flows depending on the load of the middlebox. If the middlebox can accommodate more traffic, newly arrived flows should be accepted and scheduled immediately. Otherwise, they should wait until the middlebox recovers from an over-load situation. Arbitrarily accepting newly arrived flows not only slows down the completion of already running flows, but also interferes the middlebox to tackle potential new flows. As aforementioned, if a flow misses its deadline after being processed, it indeed wastes those resources occupied by it. Such a phenomenon should be avoided because of its serious influence on other flows. Thus, if a flow has already been processed, its deadline should be guaranteed preferentially, rather than the deadlines of potential new flows. Bear this in mind, MDFS schedules flows in a

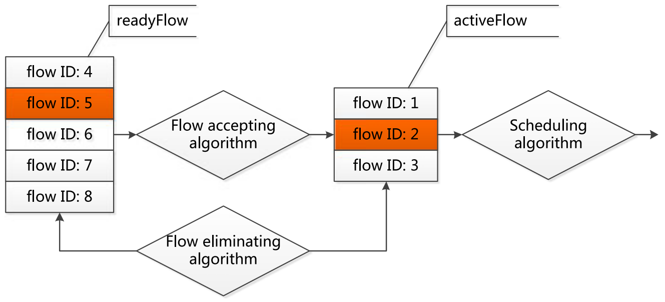


Fig. 3. MDFS scheduling framework.

non-preemptive manner, which also ensures the fairness among flows.

(2) Eliminating the flows, which cannot be completed before their deadlines. It is impossible for the scheduler to satisfy the deadline requirements of all flows. Although some flows have already been processed for a long time or some flows have just arrived, they may miss their deadlines even allocating all resources to them. If flows have already expired their deadlines before being completely processed, they should not be scheduled any more. Any flow should be eliminated immediately for avoiding unnecessary waste of resources if its deadline has expired.

According to such rationales, we design MDFS and discuss how to combine such rationales within the implementation of MDFS in Section 4.

4 MDFS Scheduling Scheme

We introduce the scheduling framework of MDFS in Section 4.1. Subsequently, we analyze the properties of MDFS in Section 4.2.

4.1 MDFS Scheduling Framework

According to the design rationales mentioned in Section 3.3, our MDFS scheduling scheme consists of three algorithms, including the flow accepting algorithm, the scheduling algorithm, and the flow eliminating algorithm. By using a precise prediction strategy, the flow accepting algorithm judges whether a flow should be scheduled, so as to tolerate the burst traffic. The scheduling algorithm is aware of the deadline of each flow and schedules packets according to their allocated timestamps. The flow eliminating algorithm is designed to alleviate the load of the middlebox. To ease the presentation, we illustrate the scheduling framework of MDFS in Fig. 3.

After receiving flows, they will not be scheduled immediately while be added into the set of ready flows, denoted as $readyFlow$. The flow accepting algorithm then judges whether the flows in $readyFlow$ should be scheduled under some specified constraints. If so, such flows will be migrated into $activeFlow$, the set of active flows. Subsequently, the scheduling algorithm decides the processing sequence of all flows in $activeFlow$. The flow eliminating algorithm always works on the two sets of flows, $readyFlow$ and $activeFlow$, so as to eliminate those unqualified flows from them. We explain details of such algorithms as follows.

Algorithm 1. The Flow Accepting Algorithm

Require: $readyFlow$ and $activeFlow$, the set of ready flows and the set of active flows, respectively. AT, the real time. VT, the maximal timestamp of packets being processed. eTime, the estimated completion time for flows in $activeFlow$ and the newly accepted flow. d_k, the deadline of f_k. s_j, the status denotation of flow f_j.

```
1:  while $f_i$ arrives do
2:    Add $f_i$ into $readyFlow$;
3:    $s_i \leftarrow 0$;
4:  for each $f_j \in readyFlow$ do
5:    $minDeadline \leftarrow MAXIMUM$;
6:    for each $f_k \in activeFlow$ do
7:      if $d_k < minDeadline$ then
8:        $minDeadline \leftarrow d_k$;
9:    $T_j \leftarrow (d_j - VT)/z_j$;
10:   if $AT < d_j$ then
11:     if $Max(L(p_j^k,1), L(p_j^k,2), \ldots, L(p_j^k,n)) \leq T_j$ then
12:       if $eTime \leq Min(minDeadline, d_j)$ then
13:         Add $f_j$ into $activeFlow$;
14:         $S(p_j^1,1) \leftarrow VT$;
15:         $s_j \leftarrow 1$;
16:         Remove $f_j$ from $readyFlow$;
```

1) The flow accepting algorithm: To tolerate the potential burst traffic, incoming flows should not be accepted arbitrarily. Algorithm1 represents the flow accepting algorithm in detail. All of flows will firstly be added into the set $readyFlow$ when arriving. Meanwhile, their status denotations will be set as 0, denoting that they have not been processed yet. A flow in $readyFlow$ will not be scheduled immediately until it is migrated into $activeFlow$. Such a migration happens only when the flow satisfies all the following constraints.

1) The flow's deadline is not expired when migrating it from the set $readyFlow$ into the set $activeFlow$.
2) The flow's transmission can be completed before its deadline if allocating all the resources to it.
3) The migration of such a flow should not influence the completion of other flows in $activeFlow$.

Obviously, the flows that have already missed their deadlines should not be migrated anymore. What's more, if a flow cannot complete its transmission even occupying all the resources, it will inevitably miss its deadline. Thus, the processing of such flows makes no sense. For avoiding unnecessary waste of resources, the processing of newly arrived flows should not influence the processing of flows, which have already been scheduled for a long time. The second and third constraints are realized through a precise prediction strategy, which can evaluate the load of the middlebox and the completion time of flows.

Before moving forward, we need to clarify two concepts: the `Actual time (AT)` and the `virtual time (VT)`. `AT` means the real time, which changes according to the realistic packet processing procedure on multiple resources. It will be utilized to judge whether a flow has already missed its deadline. Inspired by the virtual-clock [24], `VT` denotes the timestamp allocated to each packet. It indicates the time point that the packet should be scheduled, assuming that the flow is processed exclusively by the middlebox. So actually, it is used to determine the packet scheduling sequence by the scheduling algorithm.

To estimate the influence of flow f_i on other flows when it migrates into $activeFlow$, we have to predict the processing procedure of flows. Thus, we design a prediction strategy to estimate the load of the middlebox and the completion time of flows. Assuming that flow f_i has already been migrated into the set $activeFlow$ and its first packet will be allocated the maximal timestamp of packets, which are processed at that time, i.e.,

$$S(p_i^1, 1) = VT = \begin{cases} \max_{p \in P} S(p) & P \neq \phi \\ 0 & P = \phi, \end{cases} \tag{11}$$

where P denotes the set of packets being processed. $P = \phi$ means no packet is being processed right now. Such a mapping ensures that f_i will be scheduled as other flows if it is successfully migrated into $activeFlow$. Otherwise, it has to wait in $activeFlow$ even after such a migration.

As aforementioned, we relax the completion time of $p_i^{z_i}$, i.e., the last packet of f_i, to d_i. For alleviating the contention among flows on resources, the start time of its previous packets should be delayed correspondingly. That is, the remaining time before the deadline of a flow should be partitioned to all its packets. Thus, the flows with loose deadlines will not preempt resources from the flows with urgent deadlines. In MDFS, we make such subdivision in proportion to the size of packets. That is, a larger packet will be allocated a larger time slice, and vice versa. We derive the following equation to enable such a subdivision

$$\frac{T_i^k}{F(p_i^{z_i}, n) - S(p_i^1, 1)} = \frac{s(p_i^k)}{\sum_{j=1}^{z_i} s(p_i^j)}, \tag{12}$$

where T_i^k denotes the time slice allocated to p_i^k. If we set $F(p_i^{z_i}, n)$ as d_i, we get T_i^k as:

$$\begin{aligned} T_i^k &= \frac{s(p_i^k)}{\sum_{j=1}^{z_i} s(p_i^j)} * (F(p_i^{z_i}, n) - S(p_i^1, 1)) \\ &= \frac{s(p_i^k)}{s(f_i)} * (d_i - S(p_i^1, 1)). \end{aligned} \tag{13}$$

Reasonably, we take T_i^k as the interval between the start time of two successive packets of flow f_i. After knowing $S(p_i^1, 1)$, the start time of its subsequent packets on the first resource is given as:

$$S(p_i^{k+1}, 1) = S(p_i^k, 1) + T_i^k, \tag{14}$$

where $1 \leq k \leq z_i - 1$. Consequently, the start time of packets from different flows will be delayed with diverse degrees. As for packets from those flows with loose deadlines and small sizes, their start time will be delayed extremely, and vice versa. That is, such packets will get less scheduling opportunities than other packets.

For realizing such a subdivision, the information about the flow size and the packet size should be known in advance. Practically, the former can be achieved, but not the latter. Even though the packet processing time on resources has a linear relationship with the packet size [20], the variable size of packet makes it impossible to precisely predict the processing procedure of flows. A feasible solution is to set the size of packets from the same flow as equal. Under this setting, T_i^k changes to be a constant parameter as:

$$T_i = T_i^k = \frac{d_i - S(p_i^1, 1)}{z_i}. \tag{15}$$

It is worth noticing that such a strategy helps MDFS to precisely predict the processing procedure of each flow in $activeFlow$. As for all the backlogged flows in $activeFlow$, the virtual start time of their packets can be got through Eq. (14) in advance. So far, MDFS can schedule packets according to their timestamps. Thus, the packet processing sequence and the completion time of each flow in $activeFlow$ can be calculated.

As in Algorithm 1, the processing rate of f_j is confined by the maximal processing time among all resources. If $Max(L(p_j^k, 1), L(p_j^k, 2), \ldots, L(p_j^k, n))$ is larger than T_j, f_j cannot be finished even occupies all resources. Meanwhile, $minDeadline$ denotes the minimal deadline of flows in $activeFlow$, and $eTime$ denotes the final completion time of all flows in $activeFlow$ after assuming that f_j has already been migrated into it. If the minimal deadline of flows in $activeFlow \cup \{f_j\}$ is larger than $eTime$, all those flows can be completely processed before their deadlines. That is, the migration of f_i will not influence the processing of other flows in $activeFlow$. Only those flows satisfying all these constraints can be migrated into $activeFlow$. As for each flow, which has just been migrated into the set $activeFlow$, the timestamp of its first packet will be set as VT, i.e., the maximal timestamp of packets being processed at that time. Meanwhile, the status denotation of such flows will be set as 1, denoting that they are qualified to be scheduled. Otherwise, they have to wait in $readyFlow$ until they satisfy all the constraints, or be removed by the eliminating algorithm.

Algorithm 2. The Scheduling Algorithm

```
Require: $activeFlow$, the set of active flows. $s_i$, the status
    denotation of flow $f_i$. $p_x^y$, the $y$th packet of flow $f_x$.
 1: repeat
 2:   while $p_i^{k+1}$ arrives do
 3:     $S(p_i^{k+1}, 1) \leftarrow S(p_i^k, 1) + T_i^k$;;
 4:   $minTimestamp \leftarrow MAXIMUM$;
 5:   for each $f_i \in activeFlow$ do
 6:     if $S(p_i^k, 1) < minTimestamp$ then
 7:       $minTimestamp \leftarrow S(p_i^k, 1)$;
 8:       $p_x^y \leftarrow p_i^k$;
 9:   Process the packet $p_x^y$;
10:   if $p_x^y$ is the last packet of $f_x$ then
11:     $s_x \leftarrow -1$;
12: until $activeFlow = \emptyset$
```

To further improve the performance, we need to consider more fine-grained transport control or additional restrictions. MDFS sets the size of packets from the same flow as the same, so as to precisely predict the final completion time of flows in $activeFlow$ after assuming migrating a new flow into this flow set. Such a prediction strategy is less efficient when the size of packets changes frequently. Nonetheless, an alternative prediction strategy can be adopted to evaluate the load of the middlebox. Practically, if most flows can complete their transmission before their deadlines within a fixed time interval, the middlebox is not saturated and newly arrived flows should be added into $activeFlow$. That is, we can take the percentage of flows meeting their deadlines as an indicator to approximately predict the load of the middlebox. Thus, MDFS does not always need to set the size of packets from the same flow as the same, especially when the middlebox is low-loaded.

2) The scheduling algorithm: As illustrated in Fig. 3, the scheduling algorithm will determine the scheduling sequence of packets deriving from the flows in $activeFlow$, the set of active flows. It schedules packets according to their allocated timestamps. In MDFS, we use the virtual start time of a packet on the first resource as its timestamp.

As aforementioned, the first packet of each flow will be allocated a timestamp by the flow accepting algorithm when migrating the flow into the set $activeFlow$. Then, the scheduling algorithm computes the timestamp of its subsequent packets through Eq. (14). Thus, the time slice, i.e., T_i, and the timestamp of the prior packet of each flow should be recorded.

The scheduling algorithm will generate a packet processing sequence for all flows in $activeFlow$, as illustrated in Algorithm 2. As for the flows in $activeFlow$, the packet with the minimal timestamp will be processed in every scheduling period. If a flow has finished its transmission successfully, its status denotation will be set as -1. Such status denotation will be utilized by the flow eliminating algorithm to remove the corresponding flow information.

3) The flow eliminating algorithm: As aforementioned, a lot of potential factors can make flows miss their deadlines, e.g., the preemption, and the burst traffic. The detection and control of such burst flows will avoid unnecessary waste of resources. Actually, such flows can be categorized as:

- The flow which misses its deadline before being scheduled, i.e., its arrival time or waiting time has already exceeded its deadline.
- The flow which misses its deadline after being scheduled for a while.

As illustrated in Fig. 3, Algorithm 3 works on two flow sets, $readyFlow$ and $activeFlow$. Expired flows, i.e., $d_i < AT$, will be removed from such two flow sets. In addition, as for those flows, which have successfully completed their transmission before their deadlines, their status denotations will be set as -1 by Algorithm 2. The information of such flows will also be removed from $activeFlow$, since it is unnecessary to record such information any more.

Algorithm 3. The Flow Eliminating Algorithm

```
Require: $readyFlow$ and $activeFlow$, the set of ready flows and
    the set of active flows, respectively. $AT$, the real time. $d_i$,
    the deadline of $f_i$. $s_j$, the status denotation of flow $f_j$.
1:  for each $f_i \in readyFlow$ do
2:    if $d_i < AT$ then
3:      Remove $f_i$ from $readyFlow$;
4:  for each $f_j \in activeFlow$ do
5:    if $d_j < AT$ or $s_j = -1$ then
6:      Remove $f_j$ from $actualFlow$;
```

It is worth noting that MDFS completely avoids the second kind of flows by using the prediction strategy.

4.2 Properties of MDFS

MDFS possesses some attractive properties as follows.

1) Flows are scheduled fairly. As for the flows in $activeFlow$, they all get the scheduling opportunity. Actually, we can replace the flows with long processing time by the flows with short processing time, so as to alleviate the load of the middlebox. Meanwhile, the number of flows meeting their deadlines will increase in the same time interval. Nonetheless, the middlebox may fall into an unstable situation if such preemption repeats continuously. In the worst case, only small number of flows can complete their transmission before their deadlines. What's more, it is not certain that the deadline information of a flow directly indicates its importance. Based on such observations, MDFS schedules flows in a non-preemptive manner, so as to ensure the fairness among flows.

2) No flows will miss their deadlines if they start to be processed. As aforementioned, if a flow misses its deadline, it will become invalid. What's more, the resources occupied by it are wasted, which in turn delays the completion of other flows. For avoiding such scenario, MDFS adopts a precise prediction strategy to ensure the processing of flows in $activeFlow$, as illustrated in Algorithm 1.

When determining whether a flow should be migrated from $readyFlow$ to $activeFlow$, such a flow will be added into $activeFlow$ temporarily. Subsequently, the completion time of flows in $activeFlow$ will be estimated. If the migration of the new flow does not influence the completion of other flows in $activeFlow$, it will be migrated into $activeFlow$. Otherwise, it should wait in $readyFlow$ until it becomes qualified. Owing to such a precise prediction strategy, all the flows in $activeFlow$ can complete their transmission before their deadlines.

5 Performance Evaluation

In this section, we evaluate the performance of MDFS. We start with the setting of the experiment and then compare the performance of MDFS with other scheduling schemes.

5.1 Experiment Setting

As measured in DCTCP [25], the size of query traffic distributes between [2, 20 KB], and the size of delay sensitive messages distributes between [100 KB, 1 MB]. We employ $1{,}000$ flows in our experiment. The packet size of such flows uniformly distributes across [2, 200 KB]. Meanwhile, we consider four kinds of functionalities of a middlebox in our experiments. They are the IPSec encryption, the statistical monitoring, the basic forwarding and the redundancy elimination, simply called IPSE, SM, BF and RE, respectively. As discussed in DRFQ [20], the processing time of packets on resources has a linear relationship with the sizes of packets. Consequently, it can be denoted as $\alpha x + \beta$, where x denotes the size of packet, α and β are two constant parameters associating with the resource type. In our experiments, we adopt the data measured in DRFQ [20]. As for flows undergoing different function

TABLE 3
The CPU Processing Time under Different Function Components

Application	CPU Time (μs)	Deadline (ms)
IPSec Encryption	0.015x + 84.5	[10, 30]
Statistical monitoring	0.0008x + 12.1	[5, 15]
Basic forwarding	0.00286x + 6.2	[5, 15]
Redundancy Elimination	0.006987x + 10.97	[5, 15]

components, the packet processing time on the CPU is listed in Table 3.

As for the flows undergoing the four kinds of functionalities, their deadlines distribute exponentially around 20, 8, 8 and 8 ms, respectively. In addition, we set an upper bound and a lower bound on the deadlines of flows, as illustrated in Table 3. Then, we set the size of each packet proportionally to the size of the flow. That is, the size of packet will be set as 200, 400, 600, 800, 1,000 and 1,200 B, respectively, for the flows with size of [2, 20 KB), [20, 40 KB), [40, 60 KB), [60, 80 KB), [80, 100 KB) and [100, 200 KB].

As aforementioned, MDFS is appropriate for the scheduling environment with any number of resources. Without loss of generality, we assume there are two kinds of resources in our experiment, including the CPU and the link bandwidth. Each packet will be sent out through the link after being processed on the CPU, obeying to the model we defined in Section 3. As for each backlogged flow, the next packet arrives before the completion of its prior packet on the first resource.

We compare MDFS with two representative flow level scheduling schemes, i.e., FCFS and EDF, and another D^3 alike scheduling scheme. FCFS schedules flows in the flow arrival sequence. EDF prefers the flows with urgent deadlines. D^3 is a deadline-driven scheduling scheme. It adjusts the sending rate of flows according to the feedback information. However, we cannot make explicit rate allocation in a middlebox, taking the multiple resources equipped in it into account. For comparing MDFS with other deadline-driven scheduling schemes, we design a D^3 alike scheduling scheme, simply called D^3-like. That is, each packet of a flow will be allocated a tag as:

$$tag = \frac{remaining\_flow\_length}{deadline - currentTime}, \qquad (16)$$

where $currentTime$ denotes the current time. Such a D^3 alike scheduling scheme always schedules the packet with the largest tag. Although D^3-like scheduling scheme is simple and utilizes the deadline information in a coarse-grained manner, its performance outperforms FCFS and EDF in some scenarios.

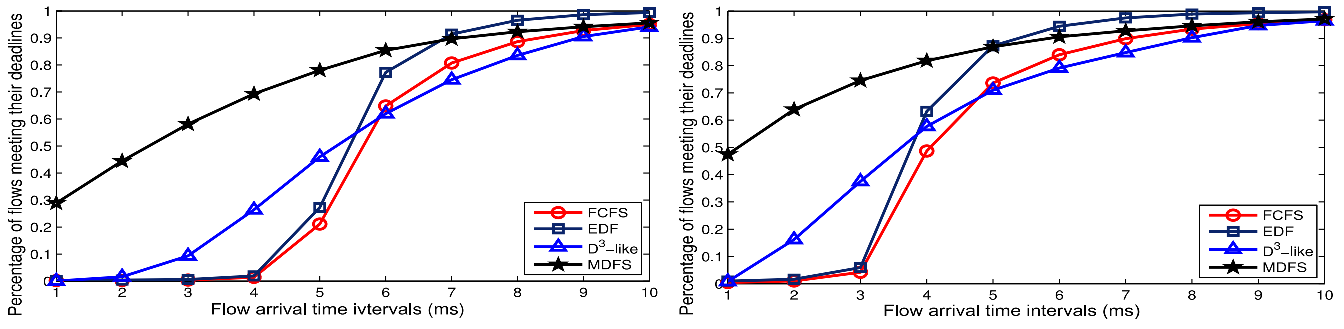
We change the sizes and deadlines of flows to generate 10 experiment configurations. All the configuration information is generated randomly, but still obeys to the bounds we mentioned previously. We conduct 10 rounds of experiments under different configurations and calculate the average value of each metric.

5.2 Availability

For evaluating the influence of the size of link bandwidth on the scheduling performance, we conduct experiments under two settings of link bandwidth, 200 and 400 Mbps. Thus, flows will be backlogged on different resources. That is, some flows need more processing time on the CPU, and other flows consume more time on the link. To evaluate the availability of the scheduling schemes, we gradually set the flow arriving time interval from 1 to 10 ms.

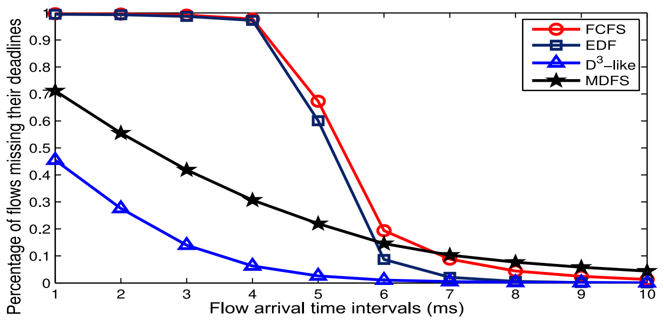
Fig. 4 reports the performance of all scheduling schemes under different flow arriving time intervals. In Fig. 4a, we set the link bandwidth as 200 Mbps. When flows arrive frequently and the middlebox cannot process all of them simultaneously, MDFS still satisfy the deadline requirements of most flows. When flows arrive every 5 ms, MDFS successfully process up to 80 percent of flows, but even EDF can only satisfy about 50 percent deadline requirements of flows, not to mention FCFS. D^3-like scheduling scheme performs commonly all the time. When setting the link bandwidth as 400 Mbps, as illustrated in Fig. 4b, the performance of all scheduling schemes grows up by a small degree. However, MDFS still outperforms other scheduling schemes when the arrival rate of flows increases, as is the case in practice.

EDF outperforms MDFS slightly under low flow traffic. The reason is that besides guaranteeing flows' deadlines, MDFS also focuses on the fairness. That is, flows deriving from different applications are treated equally, as illustrated in Figs. 6 and 7. On the contrary, only the flows with urgent deadlines are preferred by EDF. As in our experiment setting, listed in Table 3, these flows also need short processing time on resources.

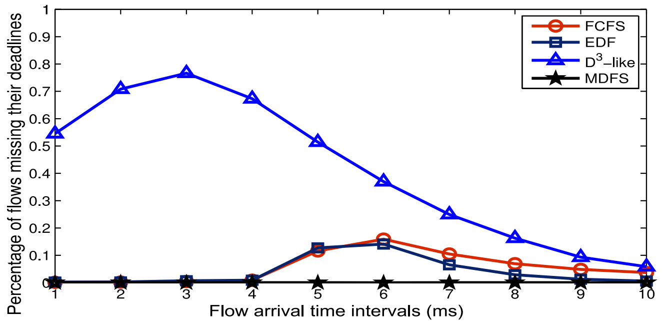


(a) Scheduling results under 200 Mbps bandwidth. (b) Scheduling results under 400 Mbps bandwidth.

Fig. 4. The performance of scheduling schemes under different time interval of arriving flows.



(a) Number of flows missing their deadlines before being scheduled.



(b) Number of flows missing their deadlines after being scheduled.

Fig. 5. Two kinds of flows missing their deadlines.

5.3 Reliability

As aforementioned, flows may miss their deadlines before or after being scheduled. The first kind of flows should be eliminated before being scheduled since it does not make any sense to process such flows. As for the second kind of flows, the resources grabbed by them are wasted and they considerably imposes impacts on the completion of other flows. We measure the number of such two kinds of flows when the time interval of arriving flows varies. Meanwhile, the link bandwidth is set as 200 Mbps.

As illustrated in Fig. 5a, when the time interval of arriving flows ranges from 1 to 4 ms, almost all flows miss their deadlines under FCFS and EDF. Consequently, simply introducing FCFS or EDF into a multi-resource environment extremely hurts the benefits of those flows with strict deadline requirements.

As for the second kind of flows in Fig. 5b, no flows miss their deadlines as long as they start to be processed by using MDFS, resorting to the precise prediction strategy. That is, MDFS provides completely reliable service for flows in $activeFlow$. The D^3-like scheduling scheme accepts flows arbitrarily, but cannot ensure the deadline requirements of all flows.

5.4 Fairness and Throughput

As for the flows undergoing different function components, we further analyze the number and throughput of flows meeting their deadlines, so as to evaluate the emphasis of different scheduling schemes on the fairness. We set the time interval of arriving flows and the link bandwidth as 6 ms and 200 Mbps, respectively.

As illustrated in Fig. 6, almost the same percentage of flows undergoing the four function components get scheduled under MDFS. That is, MDFS treats flows fairly and flows with loose deadlines also get the same scheduling opportunities as others. However, EDF prefers flows with urgent deadlines. That is, the flows, which undergo the statistical monitoring, the basic forwarding and the redundancy elimination, get more scheduling opportunities than the rest flows. The D^3-like scheduling scheme results in even more unfair scheduling consequences.

We depict the throughput of flows undergoing the four function components in Fig. 7. Compared with FCFS and EDF, flows achieve approximate throughput under MDFS. In addition, the preference of EDF extremely damages its performance when flows with long processing time are set with small deadlines.

6 Conclusion

The middleboxes with multiple types of resources realize a large range of functionalities and hence efficiently improve the network environment in data centers. Traditional deadline-driven flow scheduling schemes apply to the scheduling scenario, where just one resource is taken into account. Additionally, such schemes introduce enormous modifications on protocols and hardware, which limit the usage in practice. Moreover, when flows pass through a middlebox, existing scheduling schemes fail to guarantee the deadline requirements of flows. Such observations motivate us to propose MDFS, a deadline-driven flow scheduling scheme in a multi-resource environment, which is essential and open problem. The extensive experiment results indicate that MDFS can achieve efficient and reliable performance. Even under an overloaded setting, MDFS can treat flows fairly and satisfy the deadline requirements of most flows. Most importantly, MDFS also provides reliable service for flows such that flows will never miss their deadlines as long as they start to be processed.

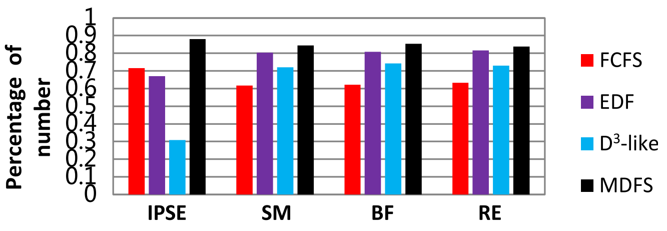


Fig. 6. The number of flows meeting their deadlines when undergoing different functionalities.

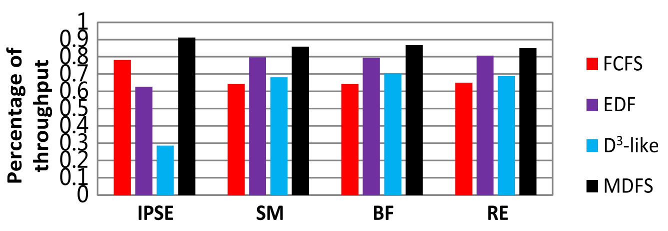


Fig. 7. The throughput of flows meeting their deadlines when undergoing different functionalities.

ACKNOWLEDGMENTS

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), the Fundamental Research Funds for the Central Universities (Grant DUT15QY20), and Prospective Research Project on Future Networks from Jiangsu Future Networks Innovation Institute. This work was done during internship at National University of Defense Technology.

REFERENCES

[1] V. Sekar, N. Egi, S. Ratnasamy, M. Reiter, and G. Shi, "Design and implementation of a consolidated middlebox architecture," in *Proc. 9th USENIX Conf. Netw. Syst. Des. Implementation*, 2012, p. 24.

[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 Conf. Appl., Technol., Archit., Protocols Comput. Commun.*, 2012, pp. 13–24.

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

[4] J. Nagle, "On packet switches with infinite storage," *IEEE Trans. Commun.*, vol. C-35, no. 4, pp. 435–438, Apr. 1987.

[5] 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, 1989.

[6] 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.

[7] S. Golestani, "A self-clocked fair queueing scheme for broadband applications," in *Proc. IEEE INFOCOM*, 1994, 636–646 .

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

[9] 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, 1993.

[10] V. Sivaraman, F. M. Chiussi, and M. Gerla, "End-to-end statistical delay service under GPS and EDF scheduling: A comparison study," in *Proc. IEEE INFOCOM*, 2001, pp. 1113–1122.

[11] N. Dukkipati, M. Kobayashi, R. Zhang-Shen, and N. McKeown, "Processor sharing flows in the internet," in *Proc. 13th Int. Conf. Quality Service*, 2005, pp. 271–285.

[12] 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, 2006.

[13] 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. 1–5.

[14] C.-Y. Hong, M. Caesar, and P. B. Godfrey, "Finishing flows quickly with preemptive scheduling," in *Proc. Conf. Appl., Technol., Archit., Protocols Comput. Commun.*, 2012, pp. 127–138.

[15] 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 Conf. Appl., Technol., Archit., Protocols Comput. Commun.*, 2012, pp. 139–150.

[16] B. Vamanan, J. Hasan, and T. Vijaykumar, "Deadline-aware datacenter TCP (D^{2}TCP)," in *Proc. ACM SIGCOMM Conf. Appl., Technol., Archit., Protocols Comput. Commun.*, 2012, pp. 115–126.

[17] D. Wischik, C. Raiciu, A. Greenhalgh, and M. Handley, "Design, implementation and evaluation of congestion control for multipath TCP," in *Proc. 8th USENIX Conf. Netw. Syst. Des. Implementation*, 2011, pp. 99–112.

[18] C. Raiciu, S. Barre, C. Pluntke, A. Greenhalgh, D. Wischik, and M. Handley, "Improving datacenter performance and robustness with multipath TCP," in *Proc. ACM SIGCOMM Conf.*, 2011, pp. 266–277.

[19] A. Ghodsi, M. Zaharia, B. Hindman, A. Konwinski, I. Stoica, and S. Shenker, "Dominant resource fairness: Fair allocation of multiple resource types," in *Proc. IEEE 8th USENIX Conf. Netw. Syst. Des. Implementation*, 2011, pp. 323–336.

[20] A. Ghodsi, V. Sekar, M. Zaharia, and I. Stoica, "Multi-resource fair queueing for packet processing," in *Proc. ACM SIGCOMM Conf. Appl., Technol., Archit., Protocols Comput. Commun.*, 2012, pp. 1–12.

[21] W. Wang, B. Liang, and B. Li, "Multi-resource generalized processor sharing for packet processing," in *Proc. ACM/IEEE 21st Int. Symp. Quality Service*, 2013, pp. 1–10.

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

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

[24] 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, 1990.

[25] M. Alizadeh, A. Greenberg, D. A. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan, "Data center TCP (DCTCP)," in *Proc. ACM SIGCOMM Conf.*, 2010, pp. 63–74.

Jianhui Zhang received the BE degree from the School of Computer Science and Technology, Harbin Engineering University, China, in 2009. He is currently working toward the PhD degree in the School of Computer Science and Technology, Dalian University of Technology, China. His research interests include datacenter networks, network protocols, and cloud computing.

Keqiu Li received the bachelor's and master's degrees from the Department of Applied Mathematics, Dalian University of Technology, in 1994 and 1997, respectively. He received the PhD degree from the Graduate School of Information Science, Japan Advanced Institute of Science and Technology, in 2005. He also has two-year postdoctoral experience at the University of Tokyo, Japan. He is currently a professor in the School of Computer Science and Technology, Dalian University of Technology, China. His research interests include internet technology, data center networks, cloud computing, and wireless networks. He is a senior member of the IEEE.

Deke Guo received the BS degree in industry engineering from Beijing University of Aeronautic and Astronautic, Beijing, China, in 2001, and the PhD 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. He is a member of the ACM and the IEEE.

Heng Qi received the bachelor's degree from Hunan University in 2004 and the master's degree from Dalian University of Technology in 2006. He received the doctorate degree from Dalian University of Technology in 2012. He was a lecture in the School of Computer Science and Technology, Dalian University of Technology, China. 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, including the *ACM Transactions on Multimedia Computing, Communications and Applications (ACM TOMCCAP)* and *Pattern Recognition (PR)*.

Wenxin Li received the BE degree from the School of Computer Science and Technology, Dalian University of Technology, China, in 2012. He is currently working toward the PhD degree in the School of Computer Science and Technology, Dalian University of Technology, China. His research interests include datacenter networks and cloud computing.

Yingwei Jin received the doctor's degree from Dalian University of Science and Technology in 2005. He is a professor in the School of Management, Dalian University of Technology, China. His research interests include computer network and security, Internet technology, and artificial intelligence. He has published more than 30 technical papers in international journals and conferences.