On Low Complexity Multi-Resource Packet Scheduling with Dominant Resource Fairness

Wei Wang, Ben Liang, and Baochun Li,

***Abstract*—Middleboxes are widely deployed in today's enterprise networks. They perform a wide range of important network functions, including WAN optimizations, intrusion detection systems, network and application level firewalls, *etc*. Depending on the processing requirement of traffic, packet processing for different traffic flows may consume vastly different amounts of hardware resources (*e.g.,* CPU and link bandwidth). Multi-resource fair queueing allows each traffic flow to receive a fair share of multiple middlebox resources. Previous schemes for multi-resource fair queueing, however, are expensive to implement at high speeds. Specifically, the time complexity to schedule a packet is $O(\log n)$, where n is the number of backlogged flows. In this paper, we design a new multi-resource fair queueing scheme that schedules packets in a manner similar to Elastic Round Robin. Our scheme requires only $O(1)$ work to schedule a packet and is simple enough to implement in practice. We show, both analytically and experimentally, that our queueing scheme achieves nearly perfect Dominant Resource Fairness.**

***Index Terms*—Middlebox, multi-resource fair queueing, dominant resource fairness, round robin.**

I. Introduction

NETWORK appliances or "middleboxes" are ubiquitous in today's networks. Recent studies report that the number of middleboxes deployed in enterprise networks is on par with the traditional L2/L3 devices [2], [3]. These middleboxes perform a variety of critical network functions, ranging from basic operations such as packet forwarding and HTTP caching to more complex processing such as WAN optimization, intrusion detection and firewalls.

As traffic through middleboxes surges [4], it is important to employ a scheduling discipline that provides *predictable service isolation* across flows. Although traditional fair queueing algorithms allow flows to receive a fair share of the output bandwidth [5], [6], [7], [8], packet scheduling in a middlebox is more complicated because flows are competing for multiple hardware resources (*e.g.,* CPU, memory bandwidth, and link bandwidth) and may have vastly different resource requirements, depending on the network functions they go through. For example, forwarding a large amount of small packets of a flow via software routers congests the memory bandwidth [9], while performing intrusion detection for external traffic is CPU intensive. Despite the heterogeneous resource requirements of traffic, flows are expected to receive predictable service isolation. This requires a *multi-resource fair queueing scheme* that makes scheduling decisions across all middlebox resources. The following properties are desired.

Fairness: The middlebox scheduler should provide some measure of service isolation to allow competing flows to have a fair share of middlebox resources. In particular, each flow should receive service (*i.e.*, throughput) at least at the level where *every resource* is equally allocated (assuming flows are equally weighted). Moreover, this service isolation should not be compromised by strategic behaviours of other flows.

Low complexity: With the ever growing line rate and the increasing volume of traffic passing through middleboxes [4], [10], it is critical to schedule packets at high speeds. This requires low time complexity when making scheduling decisions. In particular, it is desirable that this complexity is a small constant, independent of the number of traffic flows. Equally importantly, the scheduling algorithm should also be amenable to practical implementations.

While both fairness and scheduling complexity have been extensively studied for bandwidth sharing [5], [6], [7], [11], [12], multi-resource fair queueing remains a largely uncharted territory. The recent work of Ghodsi *et al.* [13] suggests a promising alternative, known as Dominant Resource Fair Queueing (DRFQ), that implements Dominant Resource Fairness (DRF) [14], [15] in the time domain. While DRFQ provides nearly perfect service isolation, it is expensive to implement. Specifically, DRFQ needs to sort *packet timestamps* [13] and requires $O(\log n)$ time complexity per packet, where n is the number of backlogged flows. With a large n, it is hard to implement DRFQ at high speeds. This problem is further aggravated by the recent middlebox innovations, where software-defined middleboxes deployed as VMs and processes are now replacing traditional network appliances with dedicated hardware [16], [17]. As more software-defined middleboxes are consolidated onto commodity and cloud servers [2], [3], a device will see an increasing amount of flows competing for multiple resources.

In this paper, we design a new packet scheduling algorithm, called Multi-Resource Round Robin (MR^3), that takes $O(1)$ time to schedule a packet, and achieves similar fairness performance as DRFQ. While *single-resource* round-robin schemes have found successful applications to fairly share the outgoing bandwidth in switches and routers [11], [18], [19], we observe that directly applying them to schedule multiple resources may lead to *arbitrary* unfairness. Instead, we first show, analytically, that nearly perfect fairness may be achieved by the simple principle of deferring the scheduling opportunity of a packet until the progress gap between two resources falls below a small threshold. We then implement

The authors are with the Department of Electrical and Computer Engineering, University of Toronto, Toronto, ON, M5S 3G4 Canada (e-mail: {weiwang, bli}@eecg.toronto.edu, liang@comm.utoronto.ca).

Part of this paper appeared in [1]. This new version contains substantial revision with additional algorithm designs, performance analyses, and simulation results.

this principle in a multi-resource packet scheduling design similar to Elastic Round Robin [19], which we show to be the most suitable variant for the middlebox environment in the design space of round robin algorithms. Both theoretical analyses and extensive simulation demonstrate that compared with DRFQ, the price we pay is a slight increase in packet latency. To our knowledge, MR^3 represents the first multi-resource fair queueing scheme that offers near-perfect fairness in $O(1)$ time. Our scheme is amenable to practical implementations, and may find a variety of applications in other multi-resource scheduling contexts such as VM scheduling inside a hypervisor.

The remainder of this paper is organized as follows. After a brief survey of the existing literature in Sec. II, we discuss the challenges of extending round-robin algorithms to the multi-resource scenario in Sec. III. Our design of MR^3 is given in Sec. IV and is evaluated via both theoretical analyses and simulation experiments in Sec. V and Sec. VI, respectively. Sec. VII concludes the paper.

II. Related Work

Unlike switches and routers where the output bandwidth is the only shared resource, middleboxes handle a variety of hardware resources and require a more complex packet scheduler. Many recent measurements, such as [9], [13], [20], report that packet processing in a middlebox may bottleneck on any of CPU, memory bandwidth, and link bandwidth, depending on the network functions applied to the traffic flow. Such a multi-resource setting significantly complicates the scheduling algorithm. As pointed out in [13], simply applying traditional fair queueing schemes [5], [6], [7], [8], [11], [21] per resource (*i.e.*, per-resource fairness) or on the bottleneck resource of the middlebox (*i.e.*, bottleneck fairness [22]) fails to offer service isolation. Furthermore, by strategically claiming some resources that are not needed, a flow may increase its service share at the price of other flows.

Ghodsi *et al.* [13] has proposed a multi-resource fair queueing algorithm, termed DRFQ, that implements DRF in the time domain, *i.e.*, it schedules packets in a way such that each flow receives roughly the same processing time on its most congested resource. It is shown to achieve service isolation and guarantee that no rational user would misreport the amount of resources it requires. Following this intuition, we have previously extended the idealized GPS model [5], [6] to Dominant Resource GPS (DRGPS), which implements strict DRF at all times [23]. By emulating DRGPS, well-known fair queueing algorithms, such as WFQ [5] and WF^{2}Q [8], can have direct extensions in the multi-resource setting. However, while all these algorithms achieve nearly perfect service isolation, they are *timestamp-based* schedulers and are expensive to implement. They all require selecting a packet with the earliest timestamp among n active flows, taking $O(\log n)$ time per packet processing. With a large n, these algorithms are hard to implement at high speeds.

Reducing the scheduling complexity is a major concern in many existing studies on high-speed networking. When there is only a single resource to schedule, round-robin schedulers [11], [18], [19] have been proposed to multiplex the output bandwidth of switches and routers, in which flows are served in circular order. These algorithms eliminate the sorting bottleneck associated with the timestamp-based schedulers, and achieve $O(1)$ time complexity per packet. Due to their extreme simplicity, round-robin schemes have been widely implemented in high-speed routers such as Cisco GSR [24].

Despite the successful application of round-robin algorithms in traditional L2/L3 devices, we observe in this work that they may lead to arbitrary unfairness when naively applied to multiple resources. Therefore, it remains unclear whether their attractiveness, *i.e.*, the implementation simplicity and low time complexity, extends to the multi-resource environment, and if it does, how a round-robin scheduler should be designed and implemented in middleboxes. We study answers to these questions in the following sections.

III. Multi-Resource Round Robin

In this section, we revisit round-robin algorithms in the traditional fair queueing literature and discuss the challenges of extending them to the multi-resource setting. We first introduce some basic concepts that will be used throughout the paper.

A. Preliminaries

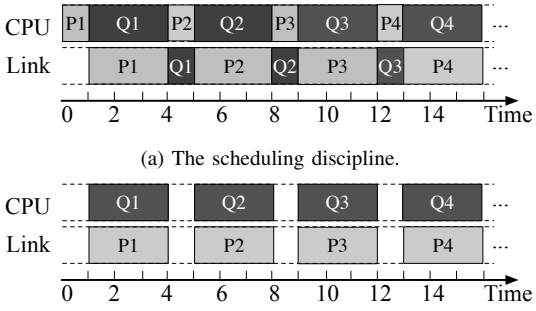
Packet Processing Time: Depending on the network functions applied to a flow, processing a packet of the flow may consume different amounts of middlebox resources. Following [13], we define the *packet processing time* as a metric for the resource requirements of a packet. Specifically, for packet p, its packet processing time on resource r, denoted $\tau_r(p)$, is defined as the time required to process the packet on resource r, normalized to the middlebox's *processing capacity* of resource r. For example, a packet may require 10 μs to process using one CPU core. A middlebox with 2 CPU cores can process 2 such packets in parallel. As a result, the packet processing time of this packet on the CPU resource of this middlebox is 5 μs.

Dominant Resource Fairness (DRF): The recently proposed Dominant Resource Fairness [14], [15] serves as a promising notion of fairness for multi-resource scheduling. Informally speaking, under DRF, any two flows receive the same processing time on the *dominant resource* of their packets in all backlogged periods. The dominant resource of a packet is the one that requires the most packet processing time. Specifically, let m be the number of resources concerned. For a packet p, its dominant resource, denoted $d(p)$, is defined as

$$d(p) = \arg\max_{1 \le r \le m} \{\tau_r(p)\}^1. \quad (1)$$

For example, consider two flows in Fig. 1a. Flow 1 sends packets P1, P2, ..., while flow 2 sends packets Q1, Q2, ... Each packet of flow 1 requires 1 time unit for CPU processing and 3 time units for link transmission, and thus

[1]Note that, in contrast to [1], [13], in this work we consider a more general scenario where the dominant resource for different packets of the same flow may be different.



(a) The scheduling discipline.

(b) The processing time received on the dominant resources.

Fig. 1. Illustration of a scheduling discipline that achieves DRF.

has the processing times $\langle 1, 3\rangle$. Each packet of flow 2, on the other hand, requires the processing times $\langle 3, 1\rangle$. In this case, the dominant resource of packets of flow 1 is link bandwidth, while the dominant resource of packets of flow 2 is CPU. We see that the scheduling scheme shown in Fig. 1a achieves DRF, under which both flows receive the same processing time on dominant resources (see Fig. 1b).

It has been shown in [13], [23] that a scheduler that achieves strict DRF at all times offers the following highly desirable properties: (1) *Predictable Service Isolation:* For each flow i, the received service (*i.e.*, throughput) is at least at the level where every resource is equally allocated. (2) *Truthfulness:* No flow can receive better service (finish faster) by misreporting the amount of resources it requires. Therefore, we adopt DRF as the notion of fairness for multi-resource scheduling.

To measure how well a packet scheduler approximates DRF, the following Relative Fairness Bound (RFB) is used as a fairness metric [13], [23]:

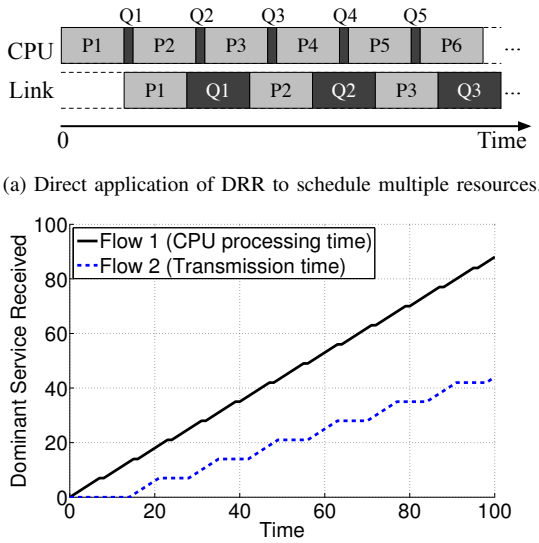
Definition 1: For any packet arrivals, let $T_i(t_1, t_2)$ be the packet processing time flow i receives on the dominant resources of its packets in the time interval (t_1, t_2). $T_i(t_1, t_2)$ is referred to as the *dominant service* flow i receives in (t_1, t_2). Let w_i be the *weight* associated with flow i, and $\mathcal{B}(t_1, t_2)$ the set of flows that are backlogged in (t_1, t_2). The *Relative Fairness Bound* (RFB) is defined as

$$\text{RFB} = \sup_{t_1,t_2;i,j\in\mathcal{B}(t_1,t_2)} \left| \frac{T_i(t_1,t_2)}{w_i} - \frac{T_j(t_1,t_2)}{w_j} \right| . \quad (2)$$

We require a scheduling scheme to have a small RFB, such that the difference between the *normalized dominant service* received by any two flows i and j, over any backlogged time period (t_1, t_2), is bounded by a small constant. For ease of presentation, we initially assume all flows are assigned the same weights, *i.e.*, $w_i = 1$ for all i. We shall discuss how this assumption can be removed in Sec. IV-C.

B. Challenges of Round-Robin Extension

As mentioned in Sec. II, among various scheduling schemes, round-robin algorithms are particularly attractive for practical implementation due to its extreme simplicity and constant time complexity. To extend it to the multi-resource setting with DRF, a natural approach is to directly apply it on the dominant resources of a flow's packets, such that in each round, flows receive roughly the same dominant services. Such



(a) Direct application of DRR to schedule multiple resources.

(b) The dominant services received by two flows.

Fig. 2. Illustration of a direct DRR extension. Each packet of flow 1 has processing times $\langle 7, 6.9\rangle$, while each packet of flow 2 has processing times $\langle 1, 7\rangle$.

a general extension can be applied to many well-known round-robin algorithms. However, such a naive extension may lead to arbitrary unfairness.

Take the well-known Deficit Round Robin (DRR) [11] as an example. When there is a single resource, DRR assigns some predefined quantum size to each flow. Each flow maintains a *deficit counter*, whose value is the currently unused transmission quota. In each round, DRR polls every backlogged flow and transmits its packets up to an amount of data equal to the sum of its quantum and deficit counter. The unused transmission quota will be carried over to the next round as the value of the flow's deficit counter. Similar to the single-resource case, one can apply DRR [11] on the dominant resources of a flow's packets as follows.

Initially, the algorithm assigns a predefined quantum size to each flow, which is also the amount of dominant service the flow is allowed to receive in one round. Each flow maintains a deficit counter that measures the currently unused portion of the allocated dominant service. Packets are scheduled in rounds, and in each round, each backlogged flow schedules as many packets as it has, as long as the dominant service consumed does not exceed the sum of its quantum and deficit counter. The unused portion of this amount is carried over to the next round as the new value of the deficit counter.

As an example, consider two flows where flow 1 sends P1, P2, ..., while flow 2 sends Q1, Q2, ... Each packet of flow 1 has processing times $\langle 7, 6.9\rangle$, *i.e.*, it requires 7 time units for CPU processing and 6.9 time units for link transmission. Each packet of flow 2 requires processing times $\langle 1, 7\rangle$. Fig. 2a illustrates the resulting schedule of the above naive DRR extension, where the quantum size assigned to both flows is 7. In round 1, both flows receive a quantum of 7, and can process 1 packet each, which consumes all the quantum awarded on the dominant resources in this round. Such a process repeats in the subsequent rounds. As a result, packets of the two flows are scheduled alternately. At the end

of each round, the received quantum is always used up, and the deficit counter remains zero.

Similar to single-resource DRR, the extension above schedules packets in $O(1)$ time[2]. However, such an extension fails to provide fair services in terms of DRF. Instead, it may lead to arbitrary unfairness with an unbounded RFB. Fig. 2b depicts the dominant services received by the two flows. We see that flow 1 receives nearly two times the dominant service flow 2 receives. With more packets being scheduled, the service gap increases, eventually leading to an unbounded RFB.

It is to be emphasized that the problem of arbitrary unfairness is not limited to DRR extension only, but generally extends to all round-robin variants. For example, one can extend Surplus Round Robin (SRR) [18] and Elastic Round Robin (ERR) [19] to the multi-resource setting in a similar way (more details can be found in Sec. IV). It is easy to verify that running the example above will give exactly the same schedule shown in Fig. 2a with an unbounded RFB[3]. In fact, due to the heterogeneous packet processing times on different resources, the work progress on one resource may be far ahead of that on the other. For example, in Fig. 2a, when CPU starts to process packet P6, the transmission of packet P3 remains unfinished. It is such a progress mismatch that leads to a significant gap between the two flows' dominant services.

In summary, directly applying round-robin algorithms on the dominant resources would fail to provide fair services. A new design is therefore required. We preview the basic idea in the next subsection.

C. Deferring the Scheduling Opportunity

The key reason that direct round-robin extensions fail is because they cannot track the flows' dominant services in real-time. Take the DRR extension as an example. In Fig. 2a, after packet Q1 is completely processed on CPU, flow 2's deficit counter is updated to 0, meaning that flow 2 has already used up the quantum allocated for dominant services (*i.e.*, link transmission) in round 1. This allows packet P2 to be processed but erroneously, as the actual consumption of this quantum incurs only when packet Q1 is transmitted on the link, after the transmission of packet P1.

To circumvent this problem, a naive fix is to withhold the scheduling opportunity of *every packet* until its previous packet is completely processed on *all resources*, which allows the scheduler to track the dominant services accurately. Fig. 3 depicts the resulting schedule when applying this fix to the DRR extension shown in Fig. 2a. We see that the difference between the dominant services received by two flows is bounded by a small constant. However, such a fairness improvement is achieved at the expense of significantly lower resource utilization. Even though multiple packets can be processed in parallel on different resources, the scheduler serves only one packet at a time, leading to poor resource utilization and high packet latency. As a result, this simple fix cannot meet the demand of high-speed networks.

[2]The $O(1)$ time complexity is conditioned on the quantum size being at least the maximum packet processing time.

[3]In either SRR or ERR extension, by scheduling 1 packet, each flow uses up all the quantum awarded in each round. As a result, packets of the two flows are scheduled alternately, the same as that in Fig. 2a.

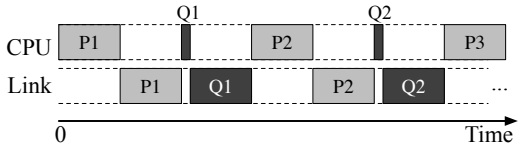
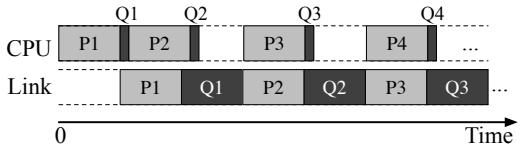
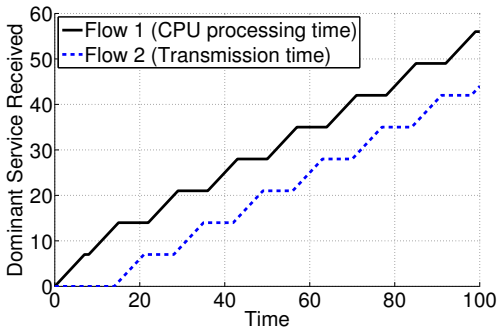


Fig. 3. Naive fix of the DRR extension shown in Fig. 2a by withholding the scheduling opportunity of every packet until its previous packet is completely processed on all resources.



(a) Schedule by MR^3.



(b) The dominant services received by two flows.

Fig. 4. Illustration of a schedule by MR^3.

To strike a balance between fairness and latency, packets should not be deferred as long as the difference of two flows' dominant services is small. This can be achieved by bounding the progress gap on different resources by a small amount. In particular, we may serve flows in rounds as follows. Whenever a packet p of a flow i is ready to be processed on the first resource (usually CPU) in round k, the scheduler checks the work progress on the last resource (usually the link bandwidth). If flow i has already received services on the last resource in the previous round $k-1$, or it has newly arrived, then packet p is scheduled immediately. Otherwise, packet p is withheld until flow i starts to receive service on the last resource in round $k-1$. As an example, Fig. 4a depicts the resulting schedule with the same input traffic as that in the example of Fig. 2a. In round 1, both packets P1 and Q1 are scheduled without delay because both flows are new arrivals. In round 2, packet P2 (resp., Q2) is also scheduled without delay, because when it is ready to be processed, flow 1 (resp., flow 2) has already started its service on the link bandwidth in round 1. In round 3, while packet P3 is ready to be processed right after packet Q2 is completely processed on CPU, it has to wait until the transmission of P2 starts. A similar process repeats for all subsequent packets.

We will show later in Sec. V that such a simple idea leads to nearly perfect fairness across flows, without incurring high packet latency. In fact, the schedule in Fig. 4a *incurs the same*

packet latency as that in Fig. 2a, but is much fairer. As we see from Fig. 4b, the difference between dominant services received by the two flows is bounded by a small constant.

IV. MR^3 Design

While the general idea introduced in the previous section is simple, implementing it as a concrete round-robin algorithm is nontrivial. We next explore the design space of round robin algorithms and identify Elastic Round Robin [19] as the most suitable variant for multi-resource fair queueing in middleboxes. The resulting algorithm is referred to as Multi-Resource Round Robin (MR^3).

A. Design Space of Round-Robin Algorithms

Many round-robin variants have been proposed in the traditional fair queueing literature. While all these variants achieve similar performance and are all feasible for the single-resource scenario, not all of them are suitable to implement the aforementioned idea in a middlebox. We investigate three typical variants, *i.e.*, Deficit Round Robin [11], Surplus Round robin [18], and Elastic Round Robin [19], and discuss their implementation issues in middleboxes as follows.

Deficit Round Robin (DRR): We have introduced the basic idea of DRR in Sec. III-B. As an analogy, one can view the behavior of each flow as maintaining a banking account. In each round, a predefined quantum is deposited into a flow's account, tracked by the deficit counter. The balance of the account (*i.e.*, the value of the deficit counter) represents the dominant service the flow is allowed to receive in the current round. Scheduling a packet is analogous to withdrawing the corresponding packet processing time on the dominant resource from the account. As long as there is sufficient balance to withdraw from the account, packet processing is allowed.

However, DRR is not amenable to implementation in middleboxes for the following two reasons. First, to ensure that a flow has sufficient account balance to schedule a packet, the processing time required on the dominant resource has to be known before packet processing. However, it is hard to know what middlebox resources are needed and how much processing time is required until the packet is processed. Also, the $O(1)$ time complexity of DRR is conditioned on the quantum size that is at least the same as the maximum packet processing time, which may not be easy to obtain in a real system. Without satisfying this condition, the time complexity could be as high as $O(N)$ [19].

Surplus Round Robin (SRR): SRR [18] allows a flow to consume more processing time on the dominant resources of its packets in one round than it has in its account. As a compensation, the excessive consumption, tracked by a *surplus counter*, will be deducted from the quantum awarded in the future rounds. In SRR, as long as the account balance (*i.e.*, surplus counter) is positive, the flow is allowed to schedule packets, and the corresponding packet processing time is withdrawn from the account after the packet is processed on its dominant resource. In this case, the packet processing time is only needed *after* the packet has been processed.

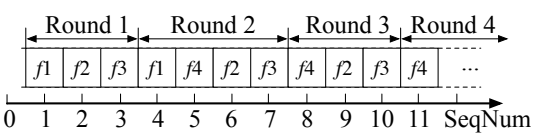


Fig. 5. Illustration of the round-robin service and the sequence number.

While SRR does not require knowing packet processing time beforehand, its $O(1)$ time remains conditioned on the predefined quantum size that is at least the same as the maximum packet processing time. Otherwise, the time complexity could be as high as $O(N)$ [19]. SRR is hence not amenable to implementation in middleboxes for the same reason mentioned above.

Elastic Round Robin (ERR): Similar to SRR, ERR [19] does not require knowing the processing time before the packet is processed. It allows flows to overdraw its permitted processing time in one round on the dominant resource, with the excessive consumption deducted from the quantum received in the next round. The difference is that instead of depositing a predefined quantum with a fixed size, in ERR, the quantum size in one round is *dynamically* set as the *maximum* excessive consumption incurred in the previous round. This ensures that each flow will *always have a positive balance* in its account at the beginning of each round, and can schedule *at least one packet*. In this case, ERR achieves $O(1)$ time complexity without knowing the maximum packet processing time *a priori*, and is the most suitable for implementation in middleboxes at high speeds.

B. MR^3 Design

While ERR serves as a promising round-robin variant for extension to the multi-resource case, there remain several challenges to implement the idea of scheduling deferral presented in Sec. III-C. How can the scheduler quickly track the work progress gap of two resources and decide when to withhold a packet? To ensure efficiency, such a progress comparison must be completed within $O(1)$ time. Note that simply comparing the numbers of packets that have been processed on two resources does not give any clue about the progress gap: due to traffic dynamics, each round may consist of different amounts of packets.

To circumvent this problem, we associate each flow i with a *sequence number SeqNum*$_i$, which is increased from 0 and is the scheduling order of the flow. We use a global variable *NextSeqNum* to record the next sequence number that will be assigned to a flow. The value of *NextSeqNum* is initialized to 0 and increased by 1 every time a flow is processed. Each flow i also records its sequence number in the previous round, tracking by *PreviousRoundSeqNum*$_i$. For example, consider Fig. 5. Initially, flows 1, 2 and 3 are backlogged and are served in sequence in round 1, with sequence numbers 1, 2 and 3, respectively. While flow 2 is being served in round 1, flow 4 becomes active. Flow 4 is therefore scheduled right after flow 1 in round 2, with a sequence number 5. After round 2, flow 1 has no packet to serve and becomes inactive. As a result, only

flows 2, 3 and 4 are serviced in round 3, where their sequence numbers in the previous round are 6, 7 and 5, respectively.

We use sequence numbers to track the work progress on a resource. Whenever a packet p is scheduled to be processed, it is stamped with its flow's sequence number. By checking the sequence number stamped to the packet that is being processed on a resource, the scheduler knows exactly the work progress on that resource.

Besides sequence numbers, the following important variables are also used in the algorithm.

Active list: The algorithm maintains an *ActiveFlowList* to track backlogged flows. Flows are served in a round-robin fashion. The algorithm always serves the flow at the head of the list, and after the service, this flow, if remaining active, will be moved to the tail of the list for service in the next round. A newly arriving flow is always appended to the tail of the list, and will be served in the next round. We also use *RoundRobinCounter* to track the number of flows that have not yet been served in the current round. Initially, *ActiveFlowList* is empty and *RoundRobinCounter* is 0.

Excess counter: Each flow i maintains an *excess counter* EC_i, recording the excessive dominant service flow i incurred in one round. The algorithm also uses two variables, *MaxEC* and *PreviousRoundMaxEC*, to track the maximum excessive consumption incurred in the current and the previous round, respectively. Initially, all these variables are set to 0.

Our algorithm, referred to as MR^3, consists of two functional modules, *PacketArrival* (Module 1), which handles packet arrival events, and *Scheduler* (Module 2), which decides which packet should be processed next.

PacketArrival: This module is invoked upon the arrival of a packet. It enqueues the packet to the input queue of the flow to which the packet belongs. If this flow is previously inactive, it is then appended to the tail of the active list and will be served in the next round. The sequence number of the flow is also updated, as shown in Module 1 (line 3 to line 5).

Module 1 MR^3 PacketArrival

```
1: Let i be the flow to which the packet belongs
2: if ActiveFlowList.Contains(i) == FALSE then
3:     PreviousRoundSeqNum_i = SeqNum_i
4:     NextSeqNum = NextSeqNum + 1
5:     SeqNum_i = NextSeqNum
6:     ActiveFlowList.AppendToTail(i)
7: end if
8: Enqueue the packet to queue i
```

Scheduler: This module decides which packet should be processed next. The scheduler first checks the value of *RoundRobinCounter* to see how many flows have not yet been served in the current round. If the value is 0, then a new round starts. The scheduler sets *RoundRobinCounter* to the length of the active list (line 3), and updates *PreviousRoundMaxEC* as the maximum excessive consumption incurred in the round that has just passed (line 4), while *MaxEC* is reset to 0 for the new round (line 5).

The scheduler then serves the flow at the head of the active list. Let flow i be such a flow. Flow i receives a quantum equal to the maximum excessive consumption incurred in the previous round, and has its account balance B_i equal to the difference between the quantum and the excess counter, *i.e.*, $B_i = PreviousRoundMaxEC - EC_i$. Since $PreviousRoundMaxEC \geq EC_i$, we have $B_i \geq 0$.

Module 2 MR^3 Scheduler

```
1: while TRUE do
2:     if RoundRobinCounter == 0 then
3:         RoundRobinCounter = ActiveFlowList.Length()
4:         PreviousRoundMaxEC = MaxEC
5:         MaxEC = 0
6:     end if
7:     Flow i = ActiveFlowList.RemoveFromHead()
8:     B_i = PreviousRoundMaxEC − EC_i
9:     while B_i ≥ 0 and QueueIsNotEmpty(i) do
10:        Let q be the packet being processed on the last resource
11:        WaitUntil(q.SeqNum ≥ PreviousRoundSeqNum_i)
12:        Packet p = Dequeue(i)
13:        p.SeqNum = SeqNum_i
14:        ProcessPacket(p)
15:        B_i = B_i − DominantProcessingTime(p)
16:    end while
17:    if QueueIsNotEmpty(i) then
18:        ActiveFlowList.AppendToTail(i)
19:        NextSeqNum = NextSeqNum + 1
20:        PreviousRoundSeqNum_i = SeqNum_i
21:        SeqNum_i = NextSeqNum
22:        EC_i = −B_i
23:    else
24:        EC_i = 0
25:    end if
26:    MaxEC = Max(MaxEC, EC_i)
27:    RoundRobinCounter = RoundRobinCounter − 1
28: end while
```

Flow i is allowed to schedule packets (if any) as long as its balance is positive (*i.e.*, $B_i \geq 0$). To ensure a small work progress gap between two resources, the scheduler keeps checking the sequence number stamped to the packet that is being processed on the last resource[4] and compares it with flow i's sequence number in the previous round. The scheduler waits until the former exceeds the latter, at which time the progress gap between any two resources is within 1 round. The scheduler then dequeues a packet from the input queue of flow i, stamps the sequence number of flow i, and performs deep packet processing on CPU, which is also the first middlebox resource required by the packet. After CPU processing, the scheduler knows exactly how the packet should be processed next and what resources are required. The packet is then pushed to the buffer of the next resource for processing. Meanwhile, the packet processing time on each resource can also be accurately estimated after CPU processing, *e.g.*, using some simple packet profiling technique introduced in [13]. The scheduler then deducts the dominant processing time of the packet from flow i's balance. The service for flow i continues until flow i has no packet to process or its balance becomes negative.

If flow i is no longer active after service in the current round, its excess counter will be reset to 0. Otherwise, flow i is appended to the tail of the active list for service in the next

[4]If no packet is being processed, we take the sequence number of the packet that has recently been processed.

round. In this case, a new sequence number is associated with flow i. The excess counter EC_i is also updated as the account deficit of flow i. Finally, before serving the next flow, the scheduler updates *MaxEC* and decreases *RoundRobinCounter* by 1, indicating that one flow has already finished service in the current round.

C. Weighted MR^3

So far, we have assumed that all flows are of the same weights and are expected to receive the same level of service. For the purpose of service differentiation, flows may be assigned different weights, based on their respective Quality of Service (QoS) requirements. A scheduling algorithm should therefore provide *weighted fairness* across flows. Specifically, the packet processing time a flow receives on the dominant resources of its packets should be in proportion to its weight, over any backlogged periods.

A minor modification to the original MR^3 is sufficient to achieve weighted fairness. In line 15 of Module 2, after packet processing, the scheduler deducts the dominant processing time of the packet, normalized to its flow's weight, from that flow's balance, *i.e.*,

$$B_i = B_i - \text{DominantProcessingTime}(p)/w_i \ , \qquad (3)$$

where w_i is the weight associated with flow i. In other words, balance B_i now tracks the normalized dominant processing time available to flow i. All the other parts of the algorithm remain the same as its unweighted version. The resulting scheduler is referred to as *weighted MR^3*.

V. Analytical Results

In this section, we analyze the performance of MR^3 by deriving its time complexity, fairness, and delay bound. We also compare MR^3 with the existing multi-resource fair queueing design, *e.g.*, DRFQ [13]. The flow weights are initially assumed equal but later generalized in Sec. V-D.

A. Complexity

MR^3 is highly efficient as compared with DRFQ. One can verify that under MR^3, at least one packet is scheduled for each flow in one round. Formally, we have

Theorem 1: MR^3 makes the scheduling decisions in $O(1)$ time per packet.

Proof: We prove the theorem by showing that both enqueuing a packet (Module 1) and scheduling a packet (Module 2) finish within $O(1)$ time.

In Module 1, determining the flow at which a new packet arrives is an $O(1)$ operation. By maintaining the per-flow state, the scheduler knows if the flow is contained in the active list (line 2) within $O(1)$ time. Also, updating the sequence number (line 3 to 5), appending the flow to the active list (line 6), and enqueueing a packet (line 8) are all of $O(1)$ time complexity.

We now analyze the time complexity of scheduling a packet. In Module 2, since the quantum deposited into each flow's account is the maximum excessive consumption incurred in the previous round, a flow will always have a positive balance at the beginning of each round, *i.e.*, $B_i \geq 0$ in line 15. As a result, at least one packet is scheduled for each flow in one round. The time complexity of scheduling a packet is therefore no more than the time complexity of all the operations performed during each service opportunity. These operations include determining the next flow to be served, removing the flow from the head of the active list and possibly adding it back at the tail, all of which are $O(1)$ operations if the active list is implemented as a linked list. Additional operations include updating the sequence number, *MaxEC*, *PreviousRoundMaxEC*, *RoundRobinCounter*, and dequeuing a packet. All of them are also executed within $O(1)$ time. ■

B. Fairness

In addition to the low scheduling complexity, MR^3 provides near-perfect fairness across flows. To see this, let EC_i^k be the excess counter of flow i after round k, and MaxEC^k the maximum EC_i^k over all flow i's. Let D_i^k be the dominant service flow i receives in round k. Also, let L_i be the maximum packet processing time of flow i across all resources. Finally, let L be the maximum packet processing time across all flows, *i.e.*, $L = \max_i\{L_i\}$. We show that the following lemmas and corollaries hold throughout the execution of MR^3. The proofs are given in Appendix A.

Lemma 1: $\text{EC}_i^k \leq L_i$ for all flow i and round k.

Corollary 1: $\text{MaxEC}^k \leq L$ for all round k.

Lemma 2: For all flow i and round k, we have

$$D_i^k = \text{MaxEC}^{k-1} - \text{EC}_i^{k-1} + \text{EC}_i^k, \qquad (4)$$

where $\text{EC}_i^0 = 0$ and $\text{MaxEC}^0 = 0$.

Corollary 2: $D_i^k \leq 2L$ for all flow i and round k.

We are now ready to bound the difference of dominant services received by two flows under MR^3. Recall that $T_i(t_1, t_2)$ denotes the dominant service flow i receives in time interval (t_1, t_2). We have the following theorem.

Theorem 2: Under MR^3, the following relationship holds for any two flows i, j that are backlogged in any time interval (t_1, t_2):

$$|T_i(t_1, t_2) - T_j(t_1, t_2)| \leq 6L.$$

Proof: Suppose at time t_1 (resp., t_2), the work progress of MR^3 on resource 1 is in round R_1 (resp., R_2). For any flow i, we upper bound $T_i(t_1, t_2)$ as follows. At time t_1, the work progress on the last resource must be *at least* in round $R_1 - 1$, as the progress gap between the first and the last resource is bounded by 1 round under MR^3. Also note that at time t_2, the progress on the last resource is *at most* in round R_2. As a result, flow i receives dominant services at most in rounds $R_1 - 1, R_1, \ldots, R_2$, *i.e.*,

$$\begin{aligned} T_i(t_1, t_2) &\leq \sum_{k=R_1-1}^{R_2} D_i^k \\ &= \sum_{k=R_1-1}^{R_2} \text{MaxEC}^{k-1} - \text{EC}_i^{R_1-2} + \text{EC}_i^{R_2} \\ &\leq \sum_{k=R_1-1}^{R_2} \text{MaxEC}^{k-1} + L \ , \end{aligned} \qquad (5)$$

where the equality is derived from Lemma 2, and the last inequality is derived from Lemma 1.

We now derive the lower bound of $T_i(t_1, t_2)$. Note that at time t_1, the algorithm has not yet started round $R_1 + 1$, while at time t_2, the algorithm must have finished all processing of round $R_2 - 2$ on the last resource. As a result, all packets that belong to rounds $R_1 + 1, \ldots, R_2 - 2$ have been processed on *all* resources within (t_1, t_2). We then have

$$\begin{aligned} T_i(t_1, t_2) &\geq \sum_{k=R_1+1}^{R_2-2} D_i^k \\ &= \sum_{k=R_1+1}^{R_2-2} \mathrm{MaxEC}^{k-1} - \mathrm{EC}_i^{R_1} + \mathrm{EC}_i^{R_2-2} \\ &\geq \sum_{k=R_1+1}^{R_2-2} \mathrm{MaxEC}^{k-1} - L \\ &\geq \sum_{k=R_1-1}^{R_2} \mathrm{MaxEC}^{k-1} - 5L \ . \end{aligned} \tag{6}$$

where the last inequality is derived from Corollary 1.

For notation simplicity, let $X = \sum_{k=R_1-1}^{R_2} \mathrm{MaxEC}^{k-1}$. By (5) and (6), we have

$$X - 5L \leq T_i(t_1, t_2) \leq X + L \ . \tag{7}$$

Note that this inequality also holds for flow j, *i.e.*,

$$X - 5L \leq T_j(t_1, t_2) \leq X + L \ . \tag{8}$$

Taking the difference between (7) and (8) leads to the statement. ∎

Corollary 3: The RFB of MR^3 is $6L$.

Based on Theorem 2 and Corollary 3, we see that MR^3 bounds the difference between (normalized) dominant services received by two backlogged flows in *any time interval* by a small constant. Note that the interval (t_1, t_2) may be arbitrarily large. MR^3 therefore achieves nearly perfect DRF across all active flows, irrespective of their traffic patterns.

We note that this is a stronger fairness guarantee than the one provided by existing multi-resource fair queueing schemes, *e.g.*, DRFQ, which require that all packets of a flow have the same dominant resource throughout each backlogged period (referred to as the *resource monotonicity assumption* in [13]).

C. Latency

In addition to complexity and fairness, latency is also an important concern for a packet scheduling algorithm. Two metrics are widely used in the fair queueing literature: *startup latency* [13], [19] and *single packet delay* [25]. The former measures how long it takes for a previously inactive flow to receive service after it becomes active, while the latter measures the latency from the time when a packet reaches the head of the input queue to the time when the scheduler finishes processing the packet on all resources.

Our analysis begins with the startup latency. In particular, let m be the number of resources concerned, and n the number of backlogged flows. We have the following theorem. The proof is given in Appendix B-A.

TABLE I
PERFORMANCE COMPARISON BETWEEN MR^3 AND DRFQ, WHERE L IS THE MAXIMUM PACKET PROCESSING TIME, m IS THE NUMBER OF RESOURCES, AND n IS THE NUMBER OF ACTIVE FLOWS.

Performance	MR^3	DRFQ
Complexity	$O(1)$	$O(\log n)$
Fairness* (RFB)	$6L$	$2L$
Startup Latency	$2(m+n-1)L$	nL
Single Packet Delay	$(4m+4n-2)L$	Unknown

* The fairness analysis of DRFQ requires the resource monotonicity assumption [13]. Under the same assumption, the RFB of MR^3 is $4L$ [1].

Theorem 3: When flows are assigned the same weights, the startup latency SL of any newly backlogged flow is bounded by

$$\mathrm{SL} \leq 2(m + n - 1)L \ . \tag{9}$$

We next derive the upper bound of the single packet delay. The proof is given in Appendix B-B.

Theorem 4: When flows are assigned the same weights, for any packet p, the single packet delay $\mathrm{SPD}(p)$ is bounded by

$$\mathrm{SPD}(p) \leq (4m + 4n - 2)L \ . \tag{10}$$

Table I summarizes the derived performance of MR^3, in comparison with DRFQ [13]. We see that MR^3 significantly reduces the scheduling complexity per packet, while providing near-perfect fairness in a more general case with arbitrary traffic patterns. The price we pay, however, is longer startup latency for newly active flows. Since the number of middlebox resources is typically much smaller than the number of active flows, *i.e.,* $m \ll n$, the startup latency bound of MR^3 is approximately two times that of DRFQ, *i.e.*,

$$2(m + n - 1)L \approx 2nL \ .$$

Also note that, since single packet delay is usually hard to analyze, no analytical delay bound is given in [13]. We shall experimentally compare the latency performance of MR^3 and DRFQ later in Sec. VI.

D. Analysis of Weighted MR^3

All our analyses presented for MR^3 can also be extended to weighted MR^3. In particular, it is easy to verify that having flows assigned uneven weights will have no impact on the scheduling complexity of the algorithm. Formally, we have

Theorem 5: Weighted MR^3 makes scheduling decisions in $O(1)$ time per packet.

Moreover, the following theorem ensures that weighted MR^3 provides near-perfect weighted fairness across flows, as the difference between the normalized dominant services received by two flows is bounded by a small constant. The proof is similar to the unweighted scenario and is briefly outlined in Appendix C-A.

Theorem 6: Under weighted MR^3, the following relationship holds for any two flows i, j that are backlogged in any

time interval (t_1, t_2):

$$\left| \frac{T_i(t_1,t_2)}{w_i} - \frac{T_j(t_1,t_2)}{w_j} \right| \leq 6 \max_{1\leq i\leq n} \frac{L_i}{w_i} \ .$$

Finally, to analyze the delay performance of weighted MR^3, let

$$W = \max_{1\leq i,j\leq n} w_i/w_j \ . \tag{11}$$

The following two theorems bound the startup latency and single packet delay for weighted MR^3, respectively. Their proofs resemble the unweighted case and are briefly described in Appendix C-B.

Theorem 7: Under weighted MR^3, the startup latency is bounded by

$$\mathrm{SL} \leq (1+W)(m+n-1)L \ .$$

Theorem 8: Under weighted MR^3, for any packet p, the single packet delay $\mathrm{SPD}(p)$ is bounded by

$$\mathrm{SPD}(p) \leq (1+W)(2m+2n-1)L \ .$$

While the delay bound might be large when flows are assigned significantly uneven weights (*i.e.*, $W \gg 1$), we shall show in the next section that most packets experience slightly less delay under weighted MR^3 than they do under DRFQ, even with very large W.

VI. Simulation Results

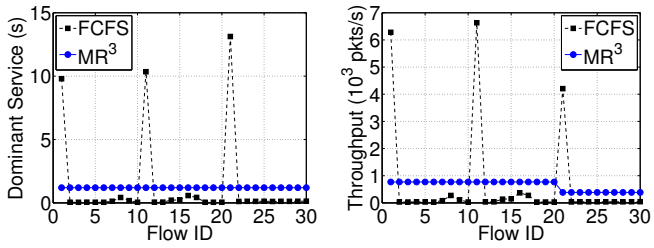
As a complementary study of the above theoretical analysis, we evaluate the performance of MR^3 via extensive simulations. First, we would like to confirm experimentally that MR^3 offers predictable service isolation and is superior to the naive *first-come-first-served* (FCFS) scheduler, as the theory indicates. Second, we want to confirm that MR^3 can quickly adapt to traffic dynamics and achieve nearly perfect DRF across flows. Third, We compare the latency performance of MR^3 with DRFQ [13] to see if the extremely low time complexity of MR^3 is achieved at the expense of significant packet delay. Fourth, we investigate how sensitive the performance of MR^3 is when packet size distributions and arrival patterns change. Finally, we evaluate the scenario where flows are assigned uneven weights, under which we compare the delay performance of weighted MR^3 with DRFQ.

A. General Setup

All simulation results are based on our event-driven packet simulator written with 3,000 lines of C++ codes. We assume resources are consumed serially, with CPU processing first, followed by link transmission. We implement 3 schedulers, FCFS, DRFQ and MR^3. The last two inspect the flows' input queues and decide which packet should be processed next. By default, packets follow Poisson arrivals, unless otherwise stated. The simulator models resource consumption of packet processing in 3 typical middlebox modules, each corresponds to one of the flow types: basic forwarding, per-flow statistical monitoring, and IPSec encryption. The first two modules are bandwidth-bound, with statistical monitoring consuming slightly more CPU resources than basic forwarding, while IPSec is CPU intensive. For direct comparison, we set the packet processing times required for each middlebox module the same as those in [13], which are based on real measurements. In particular, the CPU processing time of each module is observed to follow a simple linear model based on packet size. Table II summarizes the detailed parameters based on the measurement results reported in [13]. The link transmission time is proportional to the packet size, and the output bandwidth of the middlebox is set to 200 Mbps, the same as the experiment environment in [13]. Each of the simulation experiment spans 30 s.

TABLE II
Linear model for CPU processing time in 3 middlebox modules. Model parameters are based on the measurement results reported in [13].

Module	CPU processing time (μs)
Basic Forwarding	$0.00286 \times$ PacketSizeInBytes $+ 6.2$
Statistical Monitoring	$0.0008 \times$ PacketSizeInBytes $+ 12.1$
IPSec Encryption	$0.015 \times$ PacketSizeInBytes $+ 84.5$



(a) Dominant service received. (b) Packet throughput of flows.

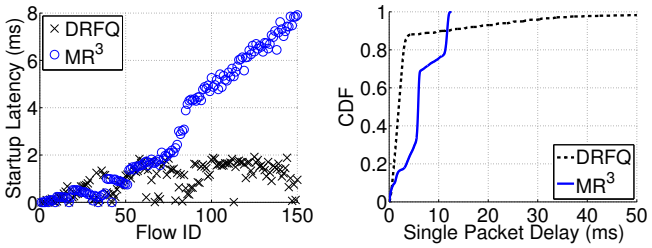
Fig. 6. Dominant services and packet throughput received by different flows under FCFS and MR^3. Flows 1, 11 and 21 are ill-behaving.

B. Service Isolation

We start off by confirming that MR^3 offers nearly perfect service isolation, which naive FCFS fails to provide. We initiate 30 flows that send 1400-byte UDP packets. Flows 1 to 10 undergo basic forwarding; 11 to 20 undergo statistical monitoring; 21 to 30 undergo IPSec encryption. We generate 3 rogue flows, *i.e.*, 1, 11 and 21, each sending 10,000 pkts/s. All other flows behaves normally, each sending 1,000 pkts/s. Fig. 6a shows the dominant services received by different flows under FCFS and MR^3. We see that under FCFS, rogue flows grab an arbitrary share of middlebox resources, while under MR^3, flows receive fair services on their dominant resources. This result is further confirmed in Fig. 6b: Under FCFS, the presence of rogue flows squeezes normal traffics to almost zero. In contrast, MR^3 ensures that all flows receive deserved, though uneven, throughput based on their dominant resource requirements, irrespective of the presence and (mis)behaviour of other traffic.

C. Latency

We next evaluate the latency penalty MR^3 pays for its extremely low time complexity, in comparison with DRFQ [13]. We implement DRFQ and measure the startup latency as well as the single packet delay of both algorithms. In particular,



(a) Startup latency of flows. (b) CDF of the single packet delay.

Fig. 7. Latency comparison between DRFQ and MR^3.

150 flows with UDP packets are initiated sequentially, where flow 1 becomes active at time 0, followed by flow 2 at time 0.2 second, and flow 3 at time 0.3 second, and so on. A flow is randomly assigned to one of the three types. To congest the middlebox resources, the packet arrival rate of each flow is set to 500 pkts/s, and the packet size is uniformly drawn from 200 bytes to 1400 bytes. Fig. 7a depicts the per-flow startup latency using both DRFQ and MR^3. Clearly, the dense and sequential flow starting times in this example represent a *worst-case scenario* for a round-robin scheduler. We see that under MR^3, flows joining the system later see larger startup latency, while under DRFQ, the startup latency is relatively consistent. This is because under MR^3, a newly active flow will have to wait for a whole round before being served. The more active flows, the more time is required to finish serving one round. As a result, the startup latency is linearly dependent on the number of active flows. While this is also true for DRFQ in the worst-case analysis (see Table I), our simulation results show that on average, the startup latency of DRFQ is smaller than MR^3. However, we see next that this advantage of DRFQ comes at the expense of highly uneven single packet delays.

Compared with the startup latency, single packet delay is a much more important delay metric. As we see from Fig. 7b, MR^3 exhibits more consistent packet delay performance, with all packets delayed less than 15 ms. In contrast, the latency distribution of DRFQ is observed to have a long tail: 90% packets are delayed less than 5 ms while the rest 10% are delayed from 5 ms to 50 ms. Further investigation reveals that these 10% packets are uniformly distributed among all flows. All results above indicate that the low time complexity and near-perfect fairness of MR^3 is achieved at the expense of only slight increase in packet latency.

D. Dynamic Allocation

We further investigate if the DRF allocation achieved by MR^3 can quickly adapt to traffic dynamics. To congest middlebox resources, we initiate 3 UDP flows each sending 20,000 1400-byte packets per second. Flow 1 undergoes basic forwarding and is active in time interval $(0, 15)$. Flow 2 undergoes statistical monitoring and is active in two intervals $(3, 10)$ and $(20, 30)$. Flow 3 undergoes IPSec encryption and is active in $(5, 25)$. The input queue of each flow can cache up to 1,000 packets. Fig. 8 shows the resource share allocated to each flow over time. Since flow 1 is bandwidth-bound and is the only active flow in $(0, 3)$, it receives 20% CPU share and

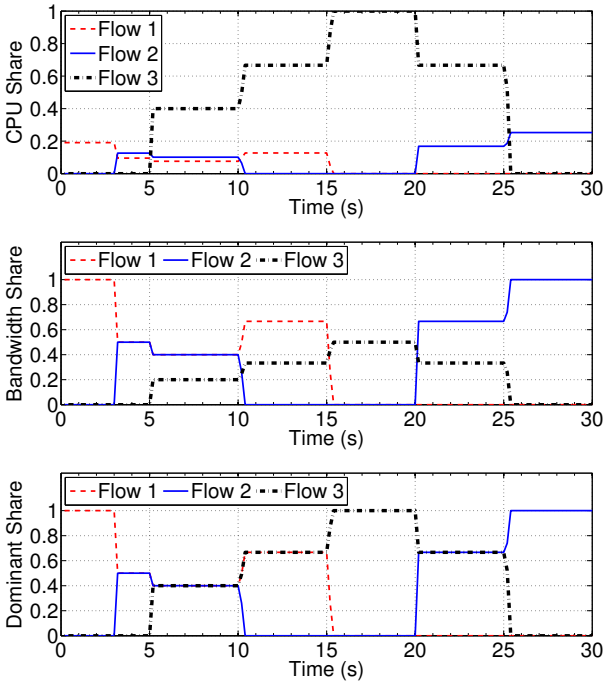
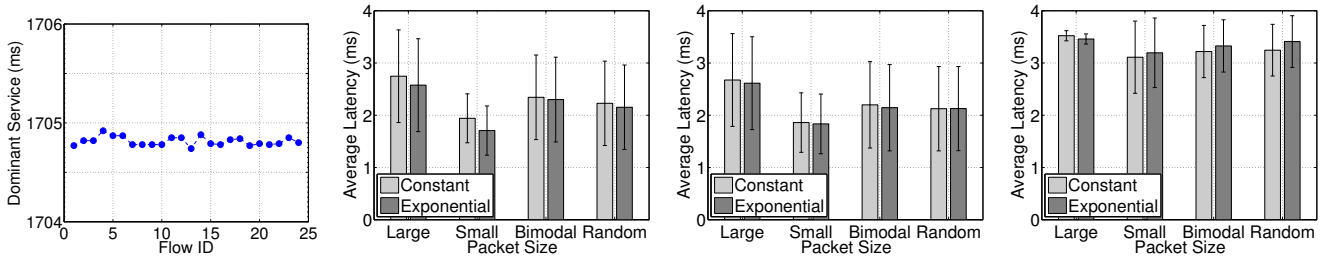


Fig. 8. MR^3 can quickly adapt to traffic dynamics and achieve DRF across all 3 flows.

all bandwidth. In $(3, 5)$, both flows 1 and 2 are active. They equally share the bandwidth on which both flows bottleneck. Later, when flow 3 becomes active at time 5, all three flows are backlogged in $(5, 10)$. Because flow 3 is CPU-bound, it grabs only 10% bandwidth share from 2 and 3, respectively, yet is allocated 40% CPU share. Similar DRF allocation is also observed in subsequent time intervals. We see that MR^3 quickly adapts to traffic dynamics, leading to nearly perfect DRF across flows.

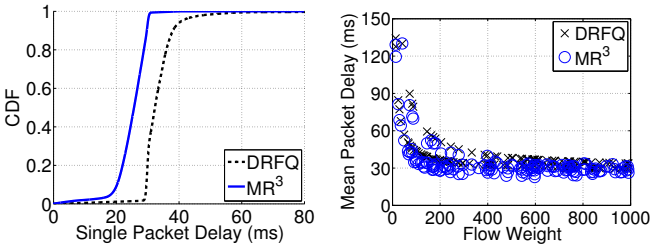
E. Sensitivity

We also evaluate the performance sensitivity of MR^3 under a mixture of different packet size distributions and arrival patterns. The simulator generates 24 UDP flows with arrival rate 10,000 pkts/s each. Flows 1 to 8 undergo basic forwarding; 9 to 16 undergo statistical monitoring; 17 to 24 undergo IPSec encryption. The 8 flows passing through the same middlebox module is further divided into 4 groups. Flows in group 1 send *large* packets with 1400 bytes; Flows in group 2 send *small* packets with 200 bytes; Flows in group 3 send *bimodal* packets that alternate between small and large; Flows in group 4 send packet with *random* size uniformly drawn from 200 bytes to 1400 bytes. Each group contains exactly 2 flows, with exponential and constant packet interarrival times, respectively. The input queue of each flow can cache up to 1,000 packets. Fig. 9a shows the dominant services received by all 24 flows, where no paritcular pattern is observed in response to distribution changes of packet sizes and arrivals. Figs. 9b, 9c and 9d show the average single packet delay observed in three middlebox modules, respectively. We find



(a) Dominant service received. (b) Basic forwarding. (c) Statistical monitoring. (d) IPSec encryption.

Fig. 9. Fairness and delay sensitivity of MR^3 in response to mixed packet sizes and arrival distributions.



(a) CDF of single packet delay. (b) Mean single packet delay.

Fig. 10. CDF and mean single packet delay under weighted MR^3.

that while the latency performance is highly consistent under different arrival patterns, it is affected by the distribution of packet size. In general, flows with small packets are slightly preferred and will see smaller latency than those with large packets. Similar preference for small-packet flows has also been observed in our experiments with DRFQ.

F. Latency with Weighted Flows

Our final experiment is to evaluate the performance of the proposed scheduler when flows are assigned uneven weights. Since we have shown in Sec. V-D that weighted MR^3 provides near-perfect fairness across flows, we focus our evaluation on the latency performance. To this end, we generate 150 UDP flows with flow weights uniformly drawn from 1 to 1000. A flow randomly chooses one of the three middlebox modules to pass through. The flow rate is set to 500 pkts/s, and the packet sizes are uniformly drawn from 200 bytes to 1400 bytes. Fig. 10a shows the CDF of the single packet delay, under both weighted MR^3 and DRFQ. Contrary to the unweighted case, most packets experience smaller scheduling delay under weighted MR^3 than they do under DRFQ. Fig. 10b shows the mean single packet delay of a flow with respect to its weight, under weighted MR^3 and DRFQ, respectively. Again, we see that weighted MR^3 consistently leads to a smaller mean delay than DRFQ. Both experiments indicate that MR^3 is competitive with DRFQ even in terms of the scheduling delay. Moreover, since in general, flows with larger weights experience smaller delays on average, both schedulers offer service discrimination in terms of the mean scheduling delay.

VII. Concluding Remarks

The potential congestion of packet processing with respect to multiple resources in a middlebox complicates the design of packet scheduling algorithms. Previously proposed multi-resource fair queueing schemes require $O(\log n)$ complexity per packet and are expensive to implement at high speeds. In this paper, we present MR^3, a multi-resource fair queueing algorithm with $O(1)$ time complexity. MR^3 serves flows in rounds. It keeps track of the work progress on each resource and withholds the scheduling opportunity of a packet until the progress gap between any two resources falls below one round. Our theoretical analyses have indicated that MR^3 implements near-perfect DRF across flows. The price we pay is a slight increase in packet latency. We have also validated our theoretical results via extensive simulation studies. To our knowledge, MR^3 is the first multi-resource fair queueing algorithm that offers near-perfect fairness with $O(1)$ time complexity. We believe that MR^3 should be easy to implement, and may find applications in other multi-resource scheduling contexts where jobs must be scheduled as entities, *e.g.*, VM scheduling inside a hypervisor.

Appendix A
Fairness Analysis

In this section, we give detailed proofs of lemmas and corollaries that are used in the fairness analysis presented in Sec. V-B. We start by proving Lemma 1.

Proof of Lemma 1: If flow i has no packets to serve (*i.e.,* the input queue is empty) after round k, then $\mathrm{EC}_i^k = 0$ (line 24 in Module 2), and the statement holds. Otherwise, let packet p be the last packet of flow i served in round k. Let B_i' be its account balance before packet p is processed. We have

$$\mathrm{EC}_i^k = \mathit{DominantProcessingTime}(p) - B_i' \leq L_i,$$

where the inequality holds because $B_i' \geq 0$ and $\mathit{DominantProcessingTime}(p) \leq L_i$. ■

We next show Lemma 2.

Proof of Lemma 2: At the beginning of round k, flow i has an account balance $B_i = \mathrm{MaxEC}^{k-1} - \mathrm{EC}_i^{k-1}$. After round k, all this amount of processing time has been consumed on the dominant resources of the processed packets, with an excessive consumption EC_i^k. The dominant service flow i received in round k is therefore $D_i = B_i + \mathrm{EC}_i^k$, which is the RHS of (4). ■

Finally, we prove Corollary 2.

Proof of Corollary 2: By Lemma 2 and noticing that $\mathrm{EC}_i^{k-1} \geq 0$, we have:

$$D_i^k \leq \mathrm{MaxEC}^{k-1} + \mathrm{EC}_i^k \leq 2L,$$

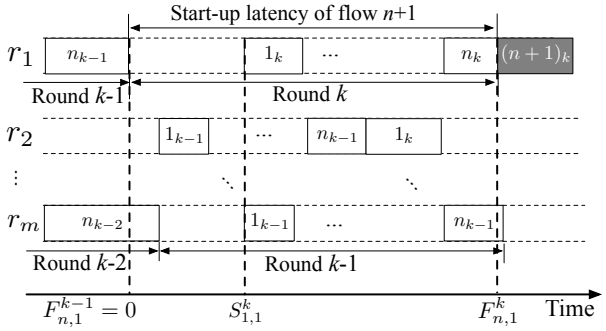


Fig. 11. Illustration of the startup latency. In the figure, flow i in round k is denoted as i_k. Flow $n+1$ becomes active when flow n has been served on resource 1 in round $k-1$, and will be served right after flow n in round k.

where the last inequality holds because of Lemma 1 and Corollary 1. ∎

APPENDIX B
LATENCY ANALYSIS

A. Analysis of Startup Latency

In this subsection, we bound the startup latency of MR^3 by proving Theorem 3.

Proof of Theorem 3: Without loss of generality, suppose n flows are backlogged in round $k-1$, with flow 1 being served first, followed by flow 2, and so on. After flow n has been served on resource 1, flow $n+1$ becomes active and is appended to the tail of the active list. Flow $n+1$ is therefore served right after flow n in round k. In particular, suppose flow $n+1$ becomes active at time 0. Let $S_{i,r}^k$ be the time when flow i starts to receive service in round k on resource r, and let $F_{i,r}^k$ be the time when the scheduler finishes serving flow i in round k on resource r. Specifically,

$$F_{n,1}^{k-1}=0\ .$$

As shown in Fig. 11, the startup latency of flow $n+1$ is

$$\mathrm{SL}=F_{n,1}^{k}\ . \tag{12}$$

The following two relationships are useful in the analysis. For all flows $i=1,\ldots,n$ and resources $r=1,\ldots,m$, we have

$$F_{i,r}^k\le S_{i,r}^k+D_i^k\le S_{i,r}^k+2L\ , \tag{13}$$

where the last inequality holds because of Corollary 2. Further, for all flows $i=1,\ldots,n$, we have

$$S_{i,1}^k=\begin{cases}\max\{S_{1,m}^{k-1},F_{n,1}^{k-1}\}, & i=1,\\ \max\{S_{i,m}^{k-1},F_{i-1,1}^{k}\}, & i=2,\ldots,n\ .\end{cases} \tag{14}$$

That is, flow i is scheduled in round k right after its previous-round service has started on the last resource m (in this case, the progress gap on two resources is no more than 1 round) and its previous flow has been served on resource 1.

The following lemma is also required in the analysis.

Lemma 3: For all flows $i=1,\ldots,n$ and all resources $r=1,\ldots,m$, we have

$$\begin{cases}S_{i,r}^{k-1}\le 2(i+r-2)L\ ,\\ F_{i,r}^{k-1}\le 2(i+r-1)L\ .\end{cases} \tag{15}$$

Proof of Lemma 3: We observe the following relationship for all resources $r=2,\ldots,m$:

$$S_{i,r}^{k-1}\le\begin{cases}\max\{F_{n,r}^{k-2},F_{1,r-1}^{k-1}\}, & i=1,\\ \max\{F_{i-1,r}^{k-1},F_{i,r-1}^{k-1}\}, & i=2,\ldots,n.\end{cases} \tag{16}$$

That is, flow i starts to receive service on resource r no later than the time when it has been served on resource $r-1$ and the time when its previous flow has been served on the same resource (see Fig. 11).

To see the statement, we apply induction to r and i. First, when $r=1$, the statement trivially holds because

$$S_{i,1}^{k-1}\le F_{i,1}^{k-1}\le F_{n,1}^{k-1}=0,\quad i=1,\ldots,n. \tag{17}$$

When $r=2, i=1$, we have

$$\begin{cases}S_{1,2}^{k-1}\le\max\{F_{n,2}^{k-2},F_{1,1}^{k-1}\}\le 2L\ ,\\ F_{1,2}^{k-1}\le S_{1,2}^{k-1}+2L\le 4L\ .\end{cases} \tag{18}$$

This is because

$$\begin{aligned}F_{n,2}^{k-2}&\le F_{n,m}^{k-2}\\ &\le S_{n,m}^{k-2}+2L && \text{(By (13))}\\ &\le S_{1,1}^{k-1}+2L && \text{(By MR}^3\text{ algorithm)}\\ &\le 2L && \text{(By (17))}\ ,\end{aligned} \tag{19}$$

and

$$S_{1,2}^{k-1}\le\max\{F_{n,2}^{k-2},F_{1,1}^{k-1}\}\le\max\{2L,0\}=2L\ . \tag{20}$$

Now assume for some r,i and $r-1,i+1$, the statement holds. Note that for $r,i+1$, we have

$$\begin{aligned}S_{i+1,r}^{k-1}&\le\max\{F_{i,r}^{k-1},F_{i+1,r-1}^{k-1}\} && \text{(By (16))}\\ &\le 2(i+r-1)L, && \text{(By induction)}\end{aligned} \tag{21}$$

and

$$F_{i+1,r}^{k-1}\le S_{i+1,r}^{k-1}+2L\le 2(i+r)L\ . \tag{22}$$

Therefore, the statement holds for $r,i=1,\ldots,n$. We then consider the case of $r+1,1$. We have

$$\begin{aligned}S_{1,r+1}^{k-1}&\le\max\{F_{n,r+1}^{k-2},F_{1,r}^{k-1}\} && \text{(By (16))}\\ &\le\max\{F_{n,m}^{k-2},2rL\}\\ &\le\max\{2L,2rL\} && \text{(By (19))}\\ &=2rL\ ,\end{aligned} \tag{23}$$

and

$$F_{1,r+1}^{k-1}\le S_{1,r+1}^{k-1}+2L\le 2(r+1)L\ . \tag{24}$$

Therefore, by induction, the statement holds. □

Lemma 3 leads to the following lemma.

Lemma 4: The following relationship holds for all flows $i=1,\ldots,n$:

$$\begin{cases}S_{i,1}^{k}\le 2(m+i-2)L\ ,\\ F_{i,1}^{k}\le 2(m+i-1)L\ .\end{cases} \tag{25}$$

Proof of Lemma 4: For $i=1$, we have

$$\begin{aligned}S_{1,1}^{k}&=\max\{S_{1,m}^{k-1},F_{n,1}^{k-1}\} && \text{(By (14))}\\ &\le\max\{2(m-1)L,0\} && \text{(By Lemma 3)}\\ &=2(m-1)L\ ,\end{aligned} \tag{26}$$

and

$$F_{1,1}^{k}\le S_{1,1}^{k}+2L\le 2mL\ . \tag{27}$$

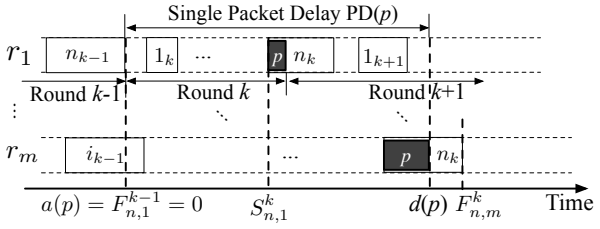


Fig. 12. Illustration of the packet latency, where flow i in round k is denoted as i_k. The figure shows the scenario under which the latency reaches its maximal value: a packet p is pushed to the top of the input queue in one round but is scheduled in the next round because of the account deficit.

Assume the statement holds for some i. Then for flow $i+1$, we have

$$\begin{aligned} S^k_{i+1,1} &= \max\{S^{k-1}_{i+1,m}, F^k_{i,1}\} && \text{(By (14))} \\ &\le 2(i+m-1)L && \text{(By Lemma 3 and (25))} \end{aligned}$$

and

$$F^k_{i+1,1} \le S^k_{i+1,1} + 2L = 2(i+m)L \ . \tag{28}$$

Hence by induction, the statement holds. □

We are now ready to prove Theorem 3. By (12), it is equivalent to show

$$\mathrm{SL} = F^k_{n,1} \le 2(m+n-1)L \ ,$$

which is a direct result of Lemma 4. ■

B. Analysis of Single Packet Delay

We next analyze the single packet delay of MR^3 by proving Theorem 4.

Proof of Theorem 4: For any packet p, let $a(p)$ be the time when packet p reaches the head of the input queue and is ready for service. Let $d(p)$ be the time when packet p has been processed on all resources and leaves the system. The single packet delay of packet p is defined as

$$\mathrm{SPD}(p) = d(p) - a(p) \ . \tag{29}$$

Without loss of generality, assume packet p belongs to flow n, and is pushed to the top of the input queue at time 0 in round $k-1$. The delay $\mathrm{SPD}(p)$ reaches its maximal value when packet p is scheduled in the next round k, as shown in Fig. 12.

We use the same notations as those in the proof of Theorem 3. Let $S^k_{i,r}$ be the time when flow i starts to receive service in round k on resource r, and let $F^k_{i,r}$ be the time when flow i has been served in round k on resource r. As shown in Fig. 12, the packet latency is

$$\mathrm{SPD}(p) = d(p) \le F^k_{n,m} \ . \tag{30}$$

We claim the following relationships for all flows $i = 1, \ldots, n$ and resources $r = 1, \ldots, m$, with which the statement holds.

$$\begin{cases} S^k_{i,r} \le 2(m+n+i+r-2)L \ , \\ F^k_{i,r} \le 2(m+n+i+r-1)L \ . \end{cases} \tag{31}$$

To see this, we extend (16) to round k by replacing $k-1$ in (16) with k:

$$S^k_{i,r} \le \begin{cases} \max\{F^{k-1}_{n,r}, F^k_{1,r-1}\}, & i = 1, \\ \max\{F^k_{i-1,r}, F^k_{i,r-1}\}, & i = 2, \ldots, n. \end{cases} \tag{32}$$

We now show (31) by induction. First, by Lemma 4, (31) holds when $r = 1$. Also, for $r = 2, i = 1$, we have

$$\begin{aligned} S^k_{1,2} &\le \max\{F^{k-1}_{n,2}, F^k_{1,1}\} && \text{(By (32))} \\ &\le \max\{2(n+1)L, 2mL\} && \text{(By (15), (25))} \\ &\le 2(m+n+1)L \ , \end{aligned}$$

and

$$F^k_{1,2} \le S^k_{1,2} + 2L = 2(m+n+2)L \ .$$

Now assume for some r, i and $r-1, i+1$, (31) holds. Note that for $r, i+1$, we have

$$\begin{aligned} S^k_{i+1,r} &\le \max\{F^k_{i,r}, F^k_{i+1,r-1}\} && \text{(By (32))} \\ &\le 2(m+n+i+r-1)L, && \text{(By induction)} \end{aligned}$$

and

$$F^k_{i+1,r} \le S^k_{i+1,r} + 2L = 2(m+n+i+r)L \ .$$

Therefore, by induction, (31) holds for $r, i = 1, \ldots, n$. We then consider the case of $r+1, 1$. We have

$$\begin{aligned} S^k_{1,r+1} &\le \max\{F^{k-1}_{n,r+1}, F^k_{1,r}\} && \text{(By (32))} \\ &\le \max\{2(n+r)L, 2(m+n+r)L\} && \text{(By Lemma 3)} \\ &= 2(m+n+r)L, \end{aligned}$$

and

$$F^k_{1,r+1} \le S^k_{1,r+1} + 2L = 2(m+n+r+1)L \ .$$

Hence by induction, (31) holds. ■

APPENDIX C
ANALYSIS OF WEIGHTED MR^3

In this section, we briefly outline how the fairness and latency analyses presented above extend to weighted MR^3.

A. Fairness Analysis of Weighted MR^3

To prove Theorem 6, we require the following relationships, which are natural extensions to those given in Sec. V-B.

Lemma 5: Under weighted MR^3, for all flow i and round k, we have

$$\mathrm{EC}^k_i \le L_i / w_i \ .$$

Corollary 4: Under weighted MR^3, for all round k, we have

$$\mathrm{MaxEC}^k \le \max_{1 \le i \le n} L_i / w_i \ .$$

Lemma 6: Under weighted MR^3, for all flow i and round k, we have

$$D^k_i / w_i = \mathrm{MaxEC}^{k-1} - \mathrm{EC}^{k-1}_i + \mathrm{EC}^k_i,$$

where $\mathrm{EC}^0_i = 0$ and $\mathrm{MaxEC}^0 = 0$.

Corollary 5: Under weighted MR^3, for all flow i and round k, the following relationship holds:

$$D^k_i \le (1+W)L \ .$$

Proof: By Lemma 6, we derive as follows

$$\begin{aligned} D_i^k &\le w_i \mathrm{MaxEC}^{k-1} + w_i \mathrm{EC}_i^k \\ &\le w_i \max_{1\le j\le n} L_j/w_j + L_i \\ &\le (1+W)L \ , \end{aligned}$$

where the second inequality is derived from Lemma 5 and Corollary 4. ∎

With these relationships, theorem 6 can be proved in a similar way as Theorem 2.

Proof sketch of Theorem 6: Following the notation and the analysis given in the proof of Theorem 2, we bound the normalized dominant service flow i receives in (t_1, t_2) as follows:

$$\sum_{k=R_1+1}^{R_2-2} \frac{D_i}{w_i} \le \frac{T_i(t_1,t_2)}{w_i} \le \sum_{k=R_1-1}^{R_2} \frac{D_i}{w_i} \ .$$

Now applying Lemma 5 and 6, we have

$$X - 5L_i/w_i \le T_i(t_1,t_2)/w_i \le X + L_i/w_i \ , \qquad (33)$$

where $X = \sum_{k=R_1-1}^{R_2} \mathrm{MaxEC}^{k-1}$. Similar inequality also holds for flow j, *i.e.*,

$$X - 5L_j/w_j \le T_j(t_1,t_2)/w_j \le X + L_j/w_j \ , \qquad (34)$$

Taking the difference between (33) and (34) leads to the statement. ∎

B. Latency Analysis of Weighted MR^3

We now briefly outline how the startup latency analysis presented in Appendix B-A extends to weighted MR^3.

Proof sketch of Theorem 7: Following the notations used in the proof of Theorem 3, it is equivalent to show

$$\mathrm{SL} = F_{n,1}^k \le (1+W)(m+n-1)L \ . \qquad (35)$$

By Corollary 5, we rewrite (13) as

$$F_{i,r}^k \le S_{i,r}^k + D_i^k \le S_{i,r}^k + (1+W)L \ . \qquad (36)$$

Combining (14), (36) and following the analysis of Lemma 3, one can easily show that for all flows $i = 1, \ldots, n$ and resources $r = 1, \ldots, m$, there must be

$$\begin{cases} S_{i,r}^{k-1} \le (1+W)(i+r-2)L \ , \\ F_{i,r}^{k-1} \le (1+W)(i+r-1)L \ . \end{cases}$$

This leads to an extension of Lemma 4, where the following relationship holds for all flows $i = 1, \ldots, n$:

$$\begin{cases} S_{i,1}^{k} \le (1+W)(m+i-2)L \ , \\ F_{i,1}^{k} \le (1+W)(m+i-1)L \ . \end{cases}$$

Taking $i = n$, we see (35) holds. ∎

Similarly, the analysis for the single packet delay also extends to weighted MR^3. In particular, one can easily extend (31) to the following inequalities, following exactly the same induction steps given in the proof of Theorem 4:

$$\begin{cases} S_{i,r}^{k} \le (1+W)(m+n+i+r-2)L \ , \\ F_{i,r}^{k} \le (1+W)(m+n+i+r-1)L \ . \end{cases} \qquad (37)$$

As a result, we have

$$\mathrm{SPD}(p) = d(p) \le F_{n,m}^k \le (1+W)(2m+2n-1) \ ,$$

which is the statement of Theorem 8.

REFERENCES

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

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

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

[4] M. Honda, Y. Nishida, C. Raiciu, A. Greenhalgh, M. Handley, and H. Tokuda, "Is it still possible to extend TCP?" in *Proc. ACM IMC*, 2011.

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

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

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

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

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

[10] Z. Wang, Z. Qian, Q. Xu, Z. Mao, and M. Zhang, "An untold story of middleboxes in cellular networks," in *Proc. SIGCOMM*, 2011.

[11] M. Shreedhar and G. Varghese, "Efficient fair queuing using deficit round-robin," *IEEE/ACM Trans. Netw.*, vol. 4, no. 3, pp. 375–385, 1996.

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

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

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

[15] D. Parkes, A. Procaccia, and N. Shah, "Beyond dominant resource fairness: Extensions, limitations, and indivisibilities," in *Proc. ACM EC*, 2012.

[16] A. Gember, P. Prabhu, Z. Ghadiyali, and A. Akella, "Toward software-defined middlebox networking," in *Proc. ACM HotNets*, 2012.

[17] J. Anderson, R. Braud, R. Kapoor, G. Porter, and A. Vahdat, "xOMB: Extensible open middleboxes with commodity servers," in *Proc. ACM/IEEE ANCS*, 2012.

[18] S. Floyd and V. Jacobson, "Link-sharing and resource management models for packet networks," *IEEE/ACM Trans. Netw.*, vol. 3, no. 4, pp. 365–386, 1995.

[19] S. Kanhere, H. Sethu, and A. Parekh, "Fair and efficient packet scheduling using elastic round robin," *IEEE Trans. Parallel Distrib. Syst.*, vol. 13, no. 3, pp. 324–336, 2002.

[20] H. Dreger, A. Feldmann, V. Paxson, and R. Sommer, "Predicting the resource consumption of network intrusion detection systems," in *Recent Advances in Intrusion Detection (RAID)*, vol. 5230. Springer, 2008, pp. 135–154.

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

[22] N. Egi, A. Greenhalgh, M. Handley, G. Iannaccone, M. Manesh, L. Mathy, and S. Ratnasamy, "Improved forwarding architecture and resource management for multi-core software routers," *Proc. IFIP NPC*, 2009.

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

[24] "Cisco GSR," http://www.cisco.com/.

[25] S. Ramabhadran and J. Pasquale, "Stratified round robin: A low complexity packet scheduler with bandwidth fairness and bounded delay," in *Proc. ACM SIGCOMM*, 2003.