Low Complexity Multi-Resource Fair Queueing with Bounded Delay

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

***Abstract*—Middleboxes are ubiquitous in today's networks. They perform deep packet processing such as content-based filtering and transformation, which requires multiple categories of resources (*e.g.*, CPU, memory bandwidth, and link bandwidth). Depending on the processing requirement of traffic, packet processing for different flows may consume vastly different amounts of resources. Multi-resource fair queueing allows flows to obtain a fair share of these resources, providing service isolation across flows. However, previous solutions for multi-resource fair queueing are either expensive to implement at high speeds, or incurring high scheduling delay for flows with uneven weights. In this paper, we present a new fair queueing algorithm, called Group Multi-Resource Round Robin (GMR^3), that schedules packets in $O(1)$ time, while achieving near-perfect fairness with a low scheduling delay bounded by a small constant. To our knowledge, it is the first provably fair, highly efficient multi-resource fair queueing algorithm with bounded delay.**

I. Introduction

Traditional Fair queueing algorithms [1], [2], [3], [4], [5], [6] are designed to schedule packets in network switches in a fair and efficient manner, and serve as the foundation of Quality of Service (QoS) research in networking. With fair queueing algorithms, a scheduler determines the order in which packets of various independent flows are forwarded on a shared output link, allocating a fair share of the outgoing bandwidth to each flow.

However, with the evolution of network appliances, output bandwidth is no longer the only shared resource in today's networks. Modern network appliances or "middleboxes" do more than just packet forwarding. In addition, they perform filtering (*e.g.,* firewalls), optimization (*e.g.,* HTTP caching and WAN optimization), and transformation (*e.g.,* dynamic request routing) based on traffic contents [7], [8], [9], which require the support of multiple resources such as CPU, memory bandwidth, and link bandwidth [10], [11]. *Multi-resource fair queueing* algorithms are therefore needed to schedule these resources to meet the QoS requirements of flows.

While fair queueing for bandwidth sharing have been extensively studied [1], [2], [3], [4], [5], [6], multi-resource fair queueing imposes new scheduling challenges as flows are competing for multiple resources and may have vastly different resource requirements. For example, flows that require forwarding a large amount of small packets congest the memory bandwidth of a software router [12], while those that require IP security encryption (IPSec) needs more CPU processing time [13]. Despite their heterogeneous resource requirements, flows are expected to receive *predictable service isolation* to meet their QoS requirements. This requires a multi-resource packet scheduler with the following three desired properties.

Fairness: The scheduler should provide some measure of service isolation across flows, so that the damaging behaviour of rogue traffic will not affect the QoS of regular flows. In particular, each flow should receive service at least at the level when *every* resource is allocated in proportion to the flow's weight, irrespective of the behaviours of other traffic.

Bounded delay: Interactive Internet applications such as video streaming and online games have stringent end-to-end delay requirements. It is hence important for a scheduler to offer a bounded scheduling delay. Such a delay bound should be a small constant, independent of the number of flows.

Low complexity: As the volume of traffic through middleboxes increases [14], [15], it is important to make scheduling decisions at high speeds. Ideally, a packet scheduler should have a time complexity that is a small constant, independent of the number of flows. In addition, the scheduling algorithm should be amenable to practical implementation.

Despite recent advances in multi-resource fair queueing (*e.g.,* [11]), how a multi-resource packet scheduler is to be designed to satisfy *all three* desirable properties remains an open and elusive challenge. Existing designs either are expensive to implement at high speeds, or provide no guaranteed delay bound. In particular, DRFQ [11], the first multi-resource fair queueing that implements Dominant Resource Fairness (DRF) [16], associates packets with timestamps, and schedules the one with the earliest timestamp. It suffers from a sorting bottleneck with high scheduling complexity, logarithmic in the number of flows. To avoid the sorting bottleneck of DRFQ, we have designed a simpler scheduler, referred to as MR^3, in our previous work [17]. MR^3 serves flows in a round-robin fashion, and reduces the scheduling complexity to $O(1)$ time per packet. However, as we shall show in Sec. II, MR^3 may incur large scheduling delays for weighted flows, and is hence unsuitable for applications with stringent delay requirements.

In this paper, we present a new packet scheduling algorithm, referred to as *Group Multi-Resource Round Robin* (GMR^3), that achieves all three desirable properties. GMR^3 groups flows with similar weights into a small number of groups, each associating with a timestamp. The scheduling decisions are made in a two-level hierarchy. At the higher level, GMR^3 makes *inter-group* scheduling decisions by choosing the group with the earliest timestamp, while at the lower level, *intra-group* scheduling serves flows within a group in a round-robin fashion. GMR^3 is highly efficient, as it requires only $O(1)$ time per packet in almost all practical scenarios. In

978-1-4799-3360-0/14/$31.00 ©2014 IEEE

978-14799-3360-0/14/$31.00 ©2014 IEEE

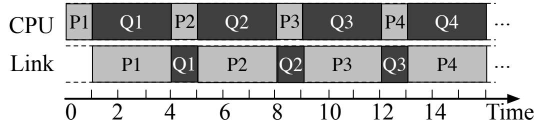


Fig. 1. A schedule that implements DRF, where flow 1 sends packets P1, P2, ..., while flow 2 sends packets Q1, Q2, ...

addition, we show that GMR^3 achieves near-perfect fairness across flows, with its scheduling delay bounded by a small constant. These desirable properties are proven analytically and validated experimentally. To our knowledge, GMR^3 is the first multi-resource fair queueing algorithm that offers near-perfect fairness with $O(1)$ time complexity and a constant scheduling delay bound. Furthermore, our algorithm may find applications in other fair scheduling contexts where multiple resources are to be time-multiplexed by different users.

The remainder of this paper is organized as follows. We clarify the design objectives and discuss the drawbacks of existing multi-resource queueing schemes in Sec. II. In Sec. III, we present our design of GMR^3 algorithm. Theoretical analysis and simulation studies are given in Sec. IV and V, respectively. Sec. VI concludes the paper.

II. Objectives and Challenges

In this section, we explain some terminologies and clarify the detailed design objectives of a multi-resource scheduler. We then briefly revisit existing multi-resource fair queueing algorithms and show that they either suffer from high complexity or incur large scheduling delays for weighted flows.

A. Design Objectives

Dominant Resource Fairness: Fairness is the primary design objective for a packet scheduler. The recently proposed Dominant Resource Fairness (DRF) serves as a promising notion of fairness in a system containing multiple resources [11], [16], [18], [19]. DRF generalizes max-min fairness to the *dominant resource* in the multi-resource setting [16]. The dominant resource is defined as the one that requires the maximum packet processing time. Specifically, let m be the number of resources under consideration. For packet p, let $\tau_r(p)$ be the time required to process it on resource r. The dominant resource r^* of packet p is $r^* = \arg\max_{1 \le r \le m} \tau_r(p)$.

Under DRF, flows receive the same processing time on their respective dominant resources (assuming flows are of equal weights). For example, consider two flows. Flow 1 requires basic forwarding, where link bandwidth is the dominant resource, while flow 2 requires security encryption, where CPU is the dominant resource. To achieve DRF, packets should be scheduled in a way such that the link transmission time flow 1 receives is equal to the CPU processing time flow 2 receives. Fig. 1 illustrates such a schedule, where flow 1 sends packets P1, P2, ..., while flow 2 sends packets Q1, Q2, ...

It has been shown in [11], [20] that a schedule that achieves DRF allows flows to receive service at least at the same level as when *every resource* is allocated in proportion to their respective weights, irrespective of the behaviours of other traffic, which is commonly known as providing *service isolation* across flows. Moreover, a DRF schedule is *work conserving* in that no resource that could be used to increase the throughput of a backlogged flow is wasted in idle. DRF hence serves as a promising notion of fairness for multi-resource fair queueing.

To measure how well a packet scheduler implements DRF, the following Relative Fairness Bound (RFB) is used as an important fairness metric [11], [17], [20].

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

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

RFB bounds the gap of dominant services received by any two flows in any backlogged period. Intuitively, the smaller the gap, the fairer the scheduler. One of our objectives is to design a scheduler with RFB being a small constant.

Scheduling Delay: In addition to fairness, scheduling delay is another important concern for a packet scheduler. The scheduling delay is defined as the time that elapses between the instant a packet reaches the head of the queue, and the instant the packet is completely processed on *all* resources. The delay is introduced by the scheduling algorithm and is also referred to as *single packet delay* in the fair queueing literature [21], [22], [23], [24]. Intuitively, flows with larger weights should experience smaller delays. Formally, for any packet p of flow i, its scheduling delay $D_i(p)$ should be within a small constant amount that is *inversely proportional* to the deserved processing weight of the flow, *i.e.,*

$$D_i(p) \le C / w_i, \tag{1}$$

where C is a constant.

Scheduling Complexity: To handle a large volume of traffic at high speeds, the scheduler must operate with low scheduling complexity, defined as the time required to make a packet scheduling decision. Ideally, this complexity should be a small constant, independent of the number of flows.

In summary, a desirable packet scheduler should offer near-perfect fairness and a constant delay bound that is inversely proportional to the flow's weight, while operating in $O(1)$ time as well. Unfortunately, none of the existing design provides all these properties, as we shall see in the next subsection.

B. Previous Work and Challenges

There are two alternative approaches that existing multi-resource fair queueing algorithms use in their designs. Timestamp-based algorithms (*e.g.,* [11], [20]) associate timestamps with each packet upon its arrival. Whenever there is a scheduling opportunity, the packet with the earliest timestamp is scheduled. Since these schedulers need to sort packet timestamps, they suffer from high scheduling complexity, requiring $O(\log n)$ time per packet, where n is the number of flows.

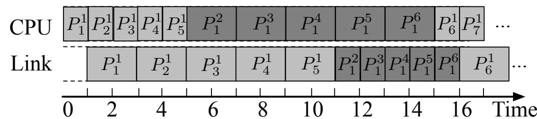


Fig. 2. MR^3 schedule fails to offer weight-proportional delay when flows are assigned uneven weights. P_k^i denotes the kth packet of flow i.

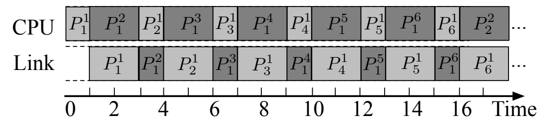


Fig. 3. An improved schedule over MR^3 in Fig. 2, where the scheduling delay is significantly reduced. P_k^i denotes the kth packet of flow i.

This sorting bottleneck significantly limits the scalability of these algorithms, and necessitates a simpler scheduler. As an alternative, Multi-Resource Round Robin (MR^3) [17] serves flows in rounds. Each flow maintains a *credit account*, and in each round, a certain amount of credit that is proportional to the flow's weight is given. The amount of available credit represents the dominant service the flow is allowed to consume in one round. A flow can overdraw its credit, and the excessive consumption will be deducted from the credit given in the next round. MR^3 eliminates the sorting bottleneck and requires only $O(1)$ complexity per packet [17].

However, MR^3 fails to offer a weight-proportional delay bound. To see this, consider an example where 6 flows are competing for both middlebox CPU and the link bandwidth. Each packet of flow 1 requires 1 time unit for CPU processing and 2 for link transmission. Each packet of other flows requires 2 time unit for CPU processing and 1 for link transmission. Flow 1 weighs 1/2, while flow 2 to 6 each weighs 1/10. The amount of credits flow 1 receives in one round is hence 5 times those given to the other flows. Fig. 2 illustrates an MR^3 schedule, where P_k^i denotes the kth packet of flow i. We see that the schedule offers weight-proportional services but not weight-proportional delay. The maximum packet scheduling delay flow 1 experiences is 13 time units (*e.g.,* packet P_6^1), more than half of that experienced by other flows (*e.g.,* packet P_2^2 has been delayed by 20 time units).

Formally, the following theorem show that MR^3 may incur large scheduling delays when flows are assigned uneven weights. The proof is deferred to our technical report [25].

Theorem 1: Under MR^3, for any flow i, the scheduling delay of its packet p is bounded by

$$D_i(p) < 4(m+W)^2 L/w_i \ ,$$

where L is the maximum packet processing time across flows, m is the number of resources under consideration, and W is the maximum ratio between weights of two flows, *i.e.,*

$$W = \max_{i,j} w_i/w_j \ . \tag{2}$$

By Theorem 1, we see that the delay bound of MR^3 critically depends on the weight distributions and may become very large when $W \gg 1$. This result has been further validated experimentally in Sec. V. To summarize, it remains open to design a multi-resource packet scheduler that offers near-perfect fairness with low complexity and a small delay bound.

Similar complexity and delay issues have also been a major challenge in the long evolution of *single-resource* fair queueing algorithms for bandwidth sharing, where both timestamp-based schemes and round robin are the two basic approaches in the design. The former provides tight delay bounds yet requires high complexity to sort packet timestamps (*e.g.,* [1], [2], [3], [6], [26]). The latter approach, on the other hand, has $O(1)$ complexity, yet incurs high scheduling delays (*e.g.,* [5], [27], [22]). To achieve the best of both worlds, one approach is to combine fairness and delay properties of timestamp-based algorithms with low time complexity of round-robin schemes. This is typically done by grouping flows into a small number of classes. The scheduler then uses the timestamp-based algorithm to determine which class to serve. Within a class, the scheduling resembles a round-robin scheme. While this strategy turns out to be an effective approach for bandwidth sharing [21], [28], [29], [23], [24], generalizing it to schedule multiple resources imposes non-trivial technical challenges. Given that flows may have different dominant resources, the scheduler has to maintain a consistent service level across all these resources. We answer this challenge in the next section.

III. Group Multi-Resource Round Robin

In this section, we present our design of Group Multi-Resource Round Robin (GMR^3) that provides all the desirable scheduling properties described in Sec. II.

A. Basic Intuition

While round robin may incur high scheduling delays in a general scenario, Theorem 1 indicates that it provides a good delay bound when flows are of similar weights. In other words, if we group flows with similar weights to a *flow group*, then within the group, round robin serves as an excellent scheduler. The challenge is to schedule *inter-group* flows with different weights.

We have observed that in MR^3, flows are always served in a "burst" mode [17]. For example, in Fig. 2, flow 1 schedules 5 packets in a row in round 1, and has to wait for an entire round to schedule its next packet P_6^1 in round 2, resulting in a long scheduling delay of that packet. Instead of serving flows in a "burst" mode, a better strategy is to spread their scheduling opportunities over time, in proportion to their respective weights. Fig. 3 illustrates an improved schedule over MR^3 in Fig. 2, where the scheduling opportunities of flow 1 are interleaved between those of other flows. Compared to the MR^3 schedule in Fig. 2, the maximum scheduling delay of flow 1 is reduced from 13 to 5, and the delay of other flows is also reduced from 20 to 16.

Our design follows exactly this intuition. The algorithm aggregates flows with similar weights into a flow group, and makes scheduling decisions in a two-level hierarchy. At the higher level, the algorithm makes inter-group scheduling decisions to determine a flow group, with the objective of distributing the scheduling opportunities over time, in proportion to the approximate weights of flows. Within a group, the *intra-group* scheduler serves flows in a round-robin fashion.

We shall show in Sec. IV that this simple combination leads to remarkable performance guarantees. For now, we focus on the detailed design in the following subsections.

B. Flow Grouping

Suppose there are n backlogged flows sharing m middlebox resources. Without loss of generality, let the flow weight w_i be normalized such that

$$\sum_{i=1}^{n} w_i = 1 \ .$$

The scheduler collects flows with similar weights into a flow group. Specifically, flow group G_k is defined as

$$G_k = \{i : 2^{-k} \leq w_i < 2^{-k+1}\}, \quad k = 1, 2, \ldots \tag{3}$$

Thus, the weights of any two flows belonging to the same flow group are within a factor of 2 of each other.

Such a grouping strategy leads to a small number of flow groups n_g, bounded by $n_g \leq \log_2 W$. As pointed out in [21], [23], [24], for a practical flow weight distribution, the number of flow groups $n_g \leq 40$ and can hence be safely assumed as a small constant. This significantly reduces the complexity of the inter-group scheduling.

C. Inter-Group Scheduling

The inter-group scheduler determines a flow group to potentially schedule a flow. Each group is associated with a timestamp, and the one with the earliest timestamp is selected. With appropriate timestamps, the scheduling opportunities of a flow group would be weight-proportionally distributed over time. Given a small number of flow groups n_g, the complexity of sorting the group timestamps is also a small constant $O(\log n_g)$. Among various timestamp-based algorithms, we find that [21] is particularly attractive for multi-resource extension, due to its simple timestamp computation. Extending other algorithms (*e.g.,* [23], [24]) to multiple resources would require referring to the idealized fluid DRGPS model [20], incurring high complexity.

The scheduler maintains an *accounting mechanism* consisting of a sequence of *virtual slots*, indexed by 0, 1, 2, ... Each slot is *exclusively* assigned to one flow, and is the scheduling opportunity of this flow. Each flow group G_k is associated with a set of *scheduling rounds* each spanning 2^k contiguous slots. The first scheduling round of flow group G_k, denoted R_1^k, starts at slot 0 and ends at slot $2^k - 1$, while the second scheduling round, denoted R_2^k, starts at slot 2^k and ends at slot $2^{k+1} - 1$, and so on. Fig. 4 gives an example. Note that the scheduling rounds of different flow groups overlap by design. The scheduler assigns each backlogged flow $i \in G_k$ exactly one slot per scheduling round of flow group G_k. This allows flow i to receive one scheduling opportunity every 2^k slots, roughly matching the flow's weight (*i.e.,* $2^{-k} \leq w_i < 2^{-k+1}$). The scheduling opportunities of flows are hence weight-proportionally distributed over time.

Following [21], a flow group is called *active* if it contains at least one backlogged flow. A backlogged flow $i \in G_k$ is called *pending* if it has not yet been assigned a slot in the current scheduling round of G_k. A flow group is called pending if it contains at least one pending flow.

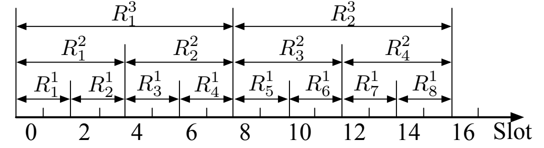


Fig. 4. An illustration of the scheduling rounds of flow groups, where R_l^k denotes the scheduling round l of flow group G_k.

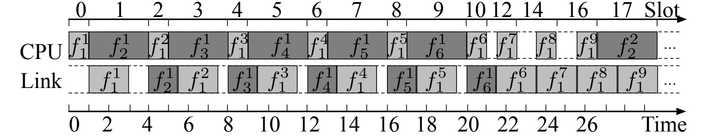


Fig. 5. An illustration of GMR^3 scheduler assigning slots to flows in the example of Fig. 2, where f_i^l denotes the packet processing for flow $i \in G_k$ in the scheduling round l of its flow group G_k. The slot axis is only for the accounting mechanism, while the time axis shows real time elapse.

For every virtual slot t, the inter-group scheduler chooses among all pending flow groups the one with the earliest timestamp, defined as *the ending slot of the current scheduling round of that flow group*. Ties are broken arbitrarily. From the selected flow group, the intra-group scheduler then chooses a pending flow and assigns it the current slot t (with details to be described in Sec. III-D). A flow temporarily ceases to be pending once it has been assigned a slot in the current scheduling round of its flow group, and will become pending again at the beginning of the next scheduling round, if it remains backlogged. If no group is pending in slot t, the slot is skipped. Algorithm 1 summarizes this inter-group scheduling process.

Algorithm 1 InterGroupScheduling

```
1:  t = 0
2:  P = {flow groups that are pending in slot 0}
3:  while TRUE do
4:      Choose G_k ∈ P, where G_k has the earliest timestamp
5:      IntraGroupScheduling(G_k)
6:      P = P − G_k if G_k is no longer pending
7:      if P = ∅ then
8:          Keep idle until there is a backlogged flow
9:          Advance t to the next slot with pending flows
10:     else
11:         t = t + 1
12:     end if
13:     P = P ∪ {flow groups that become pending in slot t}
14: end while
```

Fig. 5 illustrates an example of the inter-group scheduler assigning slots to flows in the example of Fig. 2, where f_i^l denotes the packet processing for flow i in the scheduling round l of its flow group. Flow 1 belongs to G_1 as its weight is 1/2, while flows 2 to 6 are grouped to G_4 as each of their weights is 1/10. At slot 0, both G_1 and G_4 are pending, with the end of the current scheduling round at slot 1 and slot 15, respectively. The inter-group scheduler hence picks G_1, from which the intra-group scheduler selects flow 1 as it is the only backlogged flow in G_1. Flow 1 then schedules its packets for

processing and ceases to be pending in the current scheduling round. As a result, in slot 1, only G_4 is pending and flow 2 is assigned the slot. Flow 1 becomes pending again in slot 2 as a new scheduling round of its flow group G_1 starts, and is selected for the similar reason as in slot 0. Flow groups G_1 and G_4 are hence selected alternately in the following slots until all flows of G_4 are assigned slots and cease to be pending. Note that slots 11, 13, and 15 are not shown in Fig. 5 as no flow is pending in these slots, and are hence skipped by the scheduler.

Unlike single-resource scheduling, in the multi-resource environment, a flow may not receive dominant services in its assigned slots. For example, in Fig. 5, flow 1 is assigned slot 0 in $[0, 1)$, but receives dominant services (*i.e.,* link transmission) later in $[1, 3)$. Flow 2, on the other hand, always receives dominant services (*i.e.,* CPU processing) in its assigned slots. Without appropriate control, such a potential *service asynchronicity* may lead to a significant work progress gap between two resources, resulting in poor fairness and long scheduling delay. This is also the key challenge of multi-resource scheduling as compared with its single-resource counterpart (e.g., [21], [23], [24]). We show in the next subsection that this problem is effectively addressed by the intra-group scheduler.

D. Intra-Group Scheduling

Once the flow group is determined, the intra-group scheduler chooses a pending flow from that group in a round-robin manner. Compared to round robin for bandwidth sharing (*e.g.,*, [5], [22], [27], [29], [21], [23], [24]), the intra-group scheduler operates with two important differences. First, for the purpose of DRF, the scheduler maintains a *credit system* to keep track of the dominant services a flow receives, not the amount of bits a flow transmits. Second, the scheduler employs a *progress control mechanism* to reinforce a *relatively consistent* work progress across resources, so as to eliminate the adverse effects caused by the aforementioned service asynchronicity.

Credit System: Every time a flow i is assigned a slot, it receives a credit c_i (whose size is given in (4) below), which is the time given to the flow for packet processing on its *dominant resource* in the current scheduling round. As long as there are available credits, flow i is allowed to schedule a packet for processing, and the corresponding packet processing time on the dominant resource is deducted from its total credit. A flow i can *overdraw* the processing time by scheduling *at most one more packet* than those allowed by the available credits. The excessive consumption of dominant services is tracked by the *excess counter* e_i, and will be deducted from the credit given in the next scheduling round as a penalty of overconsumption.

While MR^3 adopts a similar credit system in its design [17], the intra-group scheduler of GMR^3 operates with an important difference. Every time a flow i is assigned a slot, instead of receiving an *elastic* amount of credits in different rounds, it is given a *fixed-size* credit that is proportional to its weight w_i. Specifically, for flow $i \in G_k$, the given credit c_i is

$$c_i = 2^k L w_i \ , \tag{4}$$

where L is the maximum packet processing time. The motivation for defining credit in this manner is two-fold.

To begin with, even if two flows i, j belong to the same group G_k, flow i's weight w_i may be up to twice as large as w_j. Despite their weight difference, both flows are assigned exactly 1 slot per scheduling round of G_k. Therefore, to ensure weight-proportional dominant services, the given credits as shown in (4) are proportional to their respective weights.

Moreover, for each flow $i \in G_k$, since $2^{-k} \leq w_i < 2^{-k+1}$, the scaling factor $2^k L$ in (4) ensures that

$$L \leq c_i < 2L \ . \tag{5}$$

Because the given credits are larger than the maximum packet processing time, they can always compensate for the overconsumption of dominant service flow i incurs in the previous scheduling round. As a result, flow i will always have available credits when assigned a slot, and can schedule *at least one packet*. In addition, by (5), the given credits are roughly the same *across all flow groups*. This is significant as flow $i \in G_k$ is already assigned slots in proportion to its approximate weight 2^{-k}, so that in each slot, the scheduler should allocate all flows approximately the same dominant services.

Progress Control Mechanism: In addition to the credit system, the scheduler also employs a progress control mechanism to reinforce a *relatively consistent* processing rate *across resources*. Specifically, whenever a flow $i \in G_k$ is assigned a slot t in the scheduling round l of G_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 scheduling round $l - 1$, or flow i is a new arrival, then its packet is scheduled immediately. Otherwise, the scheduler *defers* packet scheduling *until* flow i starts to receive service on the last resource in the previous scheduling round $l-1$ of G_k. For example, as shown in Fig. 5, in slot 12, the packet processing for flow 1 (*i.e.,* f_1^7) is withheld in round 7 until the packet processed in round 6 (*i.e.,* f_1^6) starts transmission. Similar deferral has also been shown in slots 14 and 16. Intuitively, this progress control mechanism ensures that the work progress on one resource is not ahead of that on the other by more than 1 round, hence achieving an approximately consistent processing rate across resources, in spite of the potential service asynchronicity. We shall see in Sec. IV that this progress control mechanism is essential to the fairness and delay performance of GMR^3.

To summarize, Algorithm 2 gives detailed design of the intra-group scheduling. Every flow group G_k maitains an *ActiveFlowList*$[k]$ for its backlogged flows. It also uses *RoundRobinCounter*$[k]$ and *Round*$[k]$ to keep track of the current scheduling round. Every time flow group G_k is selected, the intra-group scheduler chooses flow $i \in G_k$ at the head of *ActiveFlowList*$[k]$. Flow i is given a credit to compensate for its overdraft in the previous round, and schedule packets until no credit remains or no packet is backlogged (line 6 to 15). After that, the flow ceases to be pending and is appended to the tail of the active list if it remains backlogged. Flow group G_k ceases to be pending when all its backlogged flows are serviced in the current scheduling round. If no flow is backlogged, flow group G_k becomes inactive.

Algorithm 2 IntraGroupScheduling(G_k)

```
1: if RoundRobinCounter[k] == 0 then
2:     RoundRobinCounter[k] = ActiveFlowList[k].Length()
3:     Round[k] += 1          ▷ The current scheduling round of G_k
4: end if
5: Flow i = ActiveFlowList[k].RemoveFromHead()
6: b_i = 2^k L w_i − e_i       ▷ b_i tracks the available credit of flow i
7: while IsBacklogged(i) and b_i ≥ 0 do
8:     while FlowProgressOnLastResource[i] < Round[k] − 1 do
9:         Withhold the scehduling opportunity of flow i
10:    end while
11:    Packet P = Queue[i].Dequeue()
12:    P.SchedulingRound = Round[k]
13:    ProcessPacket(P)            ▷ Schedule for CPU processing
14:    b_i = b_i − DominantProcessingTime(P)
15: end while
16: if IsBacklogged(i) then
17:    e_i = −b_i         ▷ e_i tracks the overdraft of credits of flow i
18:    ActiveFlowList[k].AppendToTail(i)
19: else
20:    e_i = 0
21: end if
22: RoundRobinCounter[k] -= 1
23: if RoundRobinCounter[k] == 0 then
24:    Flow group G_k ceases to be pending
25: end if
26: if ActiveFlowList[k] = ∅ then
27:    Deactivate(G_k)          ▷ Flow group G_k ceases to be active
28: end if
```

E. Handling New Packet Arrivals

In addition to determining the packet scheduling order, GMR^3 scheduler also needs to handle new packet arrivals. Algorithm 3 gives the detailed procedure. In addition to enqueueing the newly arrived packet p to the queue of flow $i \in G_k$ to which the packet belongs, the scheduler also appends flow i to the active list of its flow group G_k if flow i is previously inactive. Flow group G_k is also activated if it is inactive before.

Algorithm 3 PacketArrival(P)

```
1: Let i be the flow to which the newly arrived packet p belongs
2: Queue[i].Enqueue(P)
3: Let G_k be the flow group to which flow i belongs
4: if ActiveFlowList[k].Contains(i) == FALSE then
5:     ActiveFlowList[k].AppendToTail(i)
6:     if IsActive(G_k) == FALSE then
7:         Activate(G_k)           ▷ Flow group G_k becomes active
8:     end if
9: end if
```

F. Implementation and Complexity

So far, we have described the design of GMR^3. We next show that appropriate implementations allow GMR^3 to make packet scheduling decisions in $O(1)$ complexity.

Flow Grouping: To identify the flow group G_k of flow i, it suffices to locate the most significant bit of w_i that is set to 1, as $2^{-k} \leq w_i < 2^{-k+1}$. This can be accomplished in $O(1)$ by a standard *priority encoder*.

Inter-Group Scheduling: There are three important operations in Algorithm 1, *i.e.,* choosing a flow group (line 4), advancing to the earliest slot with pending groups (line 9), and updating the pending set P (line 13). Given a small number of flow groups n_g, all these operations can be accomplished in $O(1)$ time using the simple methods described in [21], which we briefly mention in the following.

The scheduler uses two bitmaps $a = a_{n_g} \dots a_2 a_1$ and $p = p_{n_g} \dots p_2 p_1$ to track the active and pending flow groups. Bit a_k is set to 1 if flow group G_k is active, and 0 otherwise. Similarly, bit p_k is 1 if group G_k is pending, and 0 otherwise.

Choosing a flow group: It is easy to check that, in all slot t, the scheduling round of flow group G_k ends earlier than those of all flow groups $G_{k'}$, where $k' > k$ (see Fig. 4). Flow group G_k hence has a higher priority to be chosen than $G_{k'}$. As a result, the chosen group G_k can be identified by locating the rightmost bit p_k of bitmap p that is set to 1. Such an operation can be done in $O(1)$ by a standard priority encoder [21].

Advancing to the earliest slot with pending groups: Because the start of the scheduling round for group G_k is also the start of a scheduling round of all groups $G_{k'}$, where $k' > k$ (see Fig. 4), the scheduler should advance to the start of the next scheduling round of the lowest-numbered flow group that is active. This can be identified by locating the rightmost bit a_k that is set to 1, and the new slot is the smallest multiple of 2^k greater than the current slot t. With the surport of priority encoder, all these operations are done in $O(1)$ time.

Updating the pending set: At slot t, an active flow group G_k becomes pending if 2^k divides t. To identify all these groups, it is sufficient to locate the least significant bit of t that is set to 1. Let it be the kth least significant bit of t. Then all active flow groups $G_{k'}$ where $k' \leq k$ become pending at t, and can be found via some simple bit operations in $O(1)$ [21].

Intra-Group Scheduling: In Algorithm 2, an essential operation is to track the work progress on the last resource of the selected flow i (line 8 to 10) to determine if the scheduling opportunity of flow i should be withheld. For the purpose of efficient implementation, a packet P of flow i, upon scheduling, is associated a tag recording the current scheduling round of flow group G_k to which flow i belongs (line 12 of Algorithm 2). Whenever packet P starts service on the last resource m, the progress of flow i on that resource is updated as the scheduling round tagged to packet P, which will be used later to determine the timing of withholding packet processing of flow i (line 8). All these operations can be done in $O(1)$.

Another operations that may introduce additional complexity is to obtain the packet processing time on the dominant resource (line 14). Note that such information is required only *after* the packet has been processed by CPU. At that time the scheduler knows exactly how the packet should be processed next and what resources are required. The packet processing time on each of the following resource can hence be accurately inferred via some simple packet profiling techniques in $O(1)$. For example, a simple linear model based on the packet size is proved to be sufficiently accurate for estimation [11].

To conclude, with appropriate implementations mentioned above, both inter-group and intra-group scheduling decisions can be made in $O(1)$ time per packet, making GMR^3 a highly efficient multi-resource scheduler for middleboxes.

IV. Performance Analysis

In this section, we analyze the properties of GMR^3 and show that it achieves near-perfect fairness with scheduling delays bounded by a small constant.

A. Fairness

For the purpose of fairness analysis, we derive the RFB of GMR^3 defined in Sec. II. We start by bounding the dominant services a flow receives in any backlogged period (t_1, t_2) as follows. The complete proof is deferred to [25].

Lemma 1: Let $T_i(t_1, t_2)$ be the dominant service a backlogged flow i receives in a time interval (t_1, t_2). We have

$$xLw_i - 9L \leq T_i(t_1, t_2) \leq xLw_i + 9L \ , \tag{6}$$

where x is the number of slots, complete and partial, that have been assigned to flows in (t_1, t_2).

Proof sketch: Let x_i be the number of slots assigned to flow $i \in G_k$ in (t_1, t_2). By Algorithm 2, the progress gap between any two resources is upper bounded by one scheduling round. It is hence easy to verify that flow i receives services on its dominant resource *at least* in $x_i - 2$ scheduling rounds, and *at most* in $x_i + 2$ scheduling rounds. The dominant services flow i receives are hence at least $(x_i - 2)c_i - L$ and at most $(x_i + 2)c_i + L$, where $c_i = 2^k Lw_i$ is the credit given to flow i, *i.e.,*

$$(x_i - 2)c_i - L \leq T_i(t_1, t_2) \leq (x_i + 2)c_i + L \ . \tag{7}$$

Also, the number of scheduling rounds of flow group G_k contained in (t_1, t_2) is at least $x_i - 2$, and is at most $x_i + 2$. Because each scheduling round of G_k spans exactly 2^k slots, we have $2^k(x_i - 2) \leq x \leq 2^k(x_i + 2)$, which is equivalent to

$$2^{-k}x - 2 \leq x_i \leq 2^{-k}x + 2 \ . \tag{8}$$

Substituting (8) to (7) and noting that $c_i \leq 2L$ by (5), we have

$$xLw_i - 9L \leq T_i(t_1, t_2) \leq xLw_i + 9L \ .$$

∎

We are now ready to derive the RFB of GMR^3 as follows.

Theorem 2: For any time interval (t_1, t_2) and any two flows i, j that are backlogged, we have

$$\left| \frac{T_i(t_1, t_2)}{w_i} - \frac{T_j(t_1, t_2)}{w_j} \right| \leq 9L \left(\frac{1}{w_i} + \frac{1}{w_j} \right) \ .$$

Proof: For any flow i, applying Lemma 1 and dividing both sides of (6) by w_i, we have

$$xL - 9L/w_i \leq T_i(t_1, t_2)/w_i \leq xL + 9L/w_i \ . \tag{9}$$

Similarly inequalities also hold for flow j, *i.e.,*

$$xL - 9L/w_j \leq T_j(t_1, t_2)/w_j \leq xL + 9L/w_j \ . \tag{10}$$

Combining (9) and (10) leads to the statement. ∎

Theorem 2 indicates that GMR^3 bounds the difference between the normalized dominant services received by two flows in *any backlogged period* by a small constant. GMR^3 hence provides near-perfect fairness across flows, irrespective of their traffic patterns. This is significant as the fairness guarantees provided by existing multi-resource fair queueing schemes, *e.g.,* [11], [17], all assume flows do not change their dominant resources throughout the backlogged periods (a.k.a., the *resource monotonicity assumption* [11]).

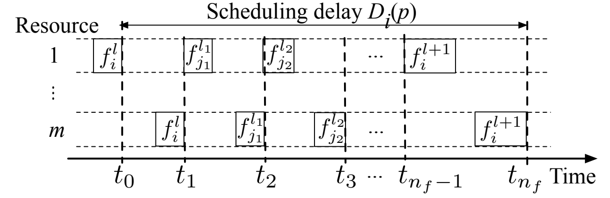


Fig. 6. The illustration of a scenario where the scheduling delay $D_i(p)$ reaches the maximum. Here, f_i^l denotes the processing of flow i in scheduling round l of its flow group.

B. Scheduling Delay

In addition to the fairness guarantees, we show that GMR^3 ensures that the scheduling delay is bounded by a small constant that is inversely proportional to the flow's weight. To see this, the following two lemmas are needed in the analysis. Their proofs are deferred to [25].

Lemma 2: Let d_i^l be the dominant services flow $i \in G_k$ receives in scheduling round l of G_k. We have

$$0 \leq d_i^l \leq 3L \ . \tag{11}$$

Lemma 3: For flow $i \in G_k$ and scheduling round l of G_k, let t_0 be the time when flow i is completely processed on resource 1 in round l of G_k, and t_1 the time when flow i is completely processed on the last resource m in round l. We have

$$t_1 - t_0 < 12mL/w_i \ .$$

We now bound the scheduling delay of GMR^3 as follows.

Theorem 3: For all flow i, the scheduling delay of its packet p is bounded by

$$D_i(p) < 24mL/w_i \ ,$$

where m is the number of resources.

Proof: For any flow $i \in G_k$, the scheduling delay of its packet p reaches its maximum when p reaches the head of the queue in scheduling round l of G_k, but is not processed until the next round $l+1$ of G_k. Since there are at most 2^{k+1} slots in between and each slot is assigned to one flow, the number of flows that have been assigned slots during this time, denoted n_f, is upper bounded by 2^{k+1}, *i.e.,* $n_f \leq 2^{k+1}$. Let these flows be $j_1, \ldots, j_{n_f}$, with their assigned slots in their current scheduling rounds $l_1, \ldots, l_{n_f}$ of their respective flow groups. In particular, $j_{n_f} = i$ and $l_{n_f} = l+1$. By Algorithm 2, flow j_1 starts service on resource 1 no later than the time its previous flow i is completely processed on the last resource m in round l of G_k. Similarly, flow j_2 starts its service on resource 1 no later than the time when its previous flow j_1 is completely processed on the last resource m in round l_1 of its flow group, and so on. Fig. 6 illustrates this scenario, where t_u is the latest time flow j_u receives service on resource 1 in round l_u of its flow group, $u = 1, 2, \ldots$. We then have

$$t_{u+1} - t_u \leq md_{j_u}^{l_u} \leq 3Lm, \quad u = 1, 2, \ldots, \tag{12}$$

TABLE I
SUMMARY OF PERFORMANCE OF GMR^3 AND EXISTING SCHEMES, WHERE n IS THE NUMBER OF FLOWS, AND m IS THE NUMBER OF RESOURCES.

Scheme	Complexity	Fairness[1]	Scheduling Delay
DRFQ [11]	$O(\log n)$	$L(1/w_i + 1/w_j)$	Unknown
MR^3 [17]	$O(1)$	$2L(1/w_i + 1/w_j)$	$4(m+W)^2 L/w_i$
GMR^3	$O(1)$	$9L(1/w_i + 1/w_j)$	$24mL/w_i$

where the second inequality is derived from Lemma 2. In other words, the time span of processing flow j_u on all resources in round l_u reaches its maximum when the processing time is maximized on every resource.

Now let t_0 be the time when packet p reaches the head of the queue in scheduling round l of its flow group, which is also the time when flow i is completely processed on resource 1 in the same round l (see Fig. 6). By Lemma 3, we have

$$t_1 - t_0 \leq 12mL/w_i \ . \tag{13}$$

With (12) and (13), we bound the scheduling delay $D_i(p)$ as follows:

$$\begin{aligned} D_i(p) &\leq \textstyle\sum_{u=1}^{n_f}(t_u - t_{u-1}) \\ &\leq 12mL/w_i + 3Lm2^{k+1} \leq 24mL/w_i \ , \end{aligned}$$

where the last inequality holds because $2^{-k} \leq w_i < 2^{-k+1}$, which implies $2^{k+1} \leq 4/w_i$. ■

Theorem 3 gives a strictly weight-proportional scheduling delay bound that is *independent of the number of flows*. This implies that a flow is guaranteed to be scheduled within a small constant amount of time that is inversely proportional to the processing rate (weight) the flow deserves, irrespective of the behaviours of other flows. To our knowledge, this is the first multi-resource packet scheduler that offers this property.

To conclude, Table I compares the performance of GMR^3 with DRFQ [11] and MR^3 [17]. We see that GMR^3 is the only scheduler that provides provably good performance guarantees on fairness, delay, and complexity.

V. SIMULATION RESULTS

For complementary study to our theoretical analysis, we experimentally evaluate the fairness and delay performance of GMR^3 via simulations.

General Setup: All simulation results are based on our event-driven packet simulator written with 3,000 lines of C++ code. Packets follow Poisson arrivals and are processed serially on resources, with CPU processing first, followed by link transmission. In addition to GMR^3, we also implement DRFQ [11] and MR^3 [17] for the purpose of comparison. The simulator simulates packet processing in 3 typical middlebox modules, *i.e.,* basic forwarding (Basic), statistical monitoring (Stat. Mon.), and IP security encryption (IPSec). The first two modules are bandwidth intensive, with monitoring consuming slightly more CPU resources, while IPSec is CPU intensive. According to the measurement results reported in [11], the CPU processing time required by each middlebox module follows a simple linear model based on packet size x, and is $\alpha_k x + \beta_k$, where α_k and β_k are parameters of module k and are summarized in Table II. The link transmission time is proportional to the packet size, and the output bandwidth of the middlebox is 200 Mbps, the same as [11].

[1]The fairness analysis of DRFQ and MR^3 requires that flows do not change their dominant resources throughout the backlogged periods [11], [17].

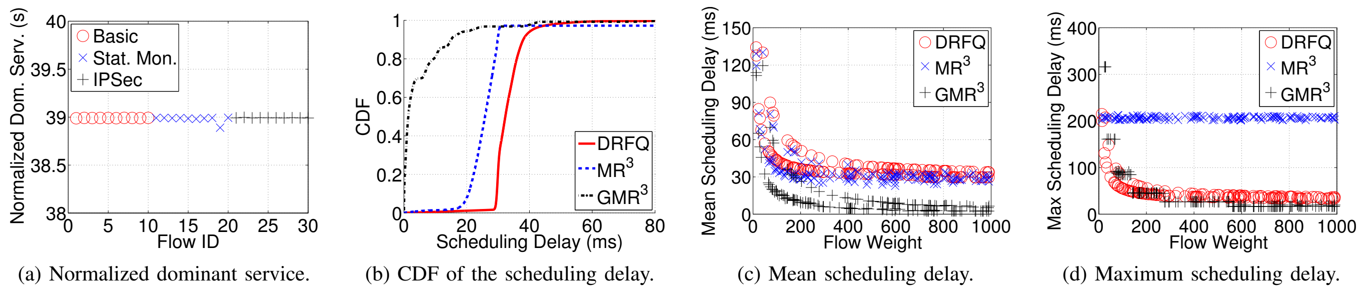
TABLE II
PARAMETERS OF LINEAR MODEL FOR CPU PROCESSING TIME IN 3 MIDDLEBOX MODULES BASED ON MEASUREMENT RESULTS IN [11].

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 $+ 84.5$

Fairness: We confirm experimentally that GMR^3 provides near-perfect service isolation across flows, irrespective of their traffic behaviours. The simulator generates 30 traffic flows that send 1300-byte UDP packets for 30 seconds. Flows 1 to 10 pass through the Basic module; flows 11 to 20 undergo statistical monitoring; while flows 21 to 30 require IPSec encryption. Among all these flows, flow 1, 11, and 21 are rogue traffic, each sending 30,000 pkts/s. All other flows behave normally, each sending 3,000 pkts/s. Flows are assigned random weights uniformly drawn from 1 to 1000. Fig. 7a depicts the dominant services, in seconds, received by different flows under GMR^3, normalized to their respective weights. We see that despite the presence of ill-behaving traffic, GMR^3 allows flows through different modules to receive weight-proportional dominant services, enforcing service isolation. Similar results have also been observed using DRFQ and MR^3, and are not shown in the figure.

Scheduling Delay: We next confirm experimentally that GMR^3 significantly improves the packet scheduling delay, as compared to existing multi-resource scheduling alternatives. The simulator generates 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. To congest the middlebox resources, the flow rate is set to 500 pkts/s, with packet sizes uniformly drawn from 200 B to 1400 B, which are the typical settings for Ethernet. For each processed packet, we record its scheduling delay, using DRFQ, MR^3, and GMR^3, respectively. The simulation spans 30 seconds.

Fig. 7b shows the CDF of the scheduling delay a packet experiences, from which we see the significance of GMR^3 on delay improvement: Using GMR^3, over 95% packets are scheduled within 20 ms, which is roughly the minimum time a packet has to wait under DRFQ and MR^3! A detailed statistics breakdown is given in Fig. 7c and 7d. Fig. 7c shows the mean scheduling delay a flow experiences with respective to its weight. We see that GMR^3 consistently leads to a smaller mean delay than the other two schedulers for almost all flows, especially for those with large weights. This delay improvement is not limited to the average case. Fig. 7d gives the maximum delay a flow experiences with respect to its weight. We see that both GMR^3 and DRFQ offer a weight-proportional delay bound. While DRFQ achieves a smaller delay bound for flows with smaller weights, GMR^3 is generally better for more important flows with medium to large weights. MR^3, on the other hand, fails to provide service differentiations among flows. Intuitively, since flows are served in rounds, in the worst case, a packet has to wait



(a) Normalized dominant service. (b) CDF of the scheduling delay. (c) Mean scheduling delay. (d) Maximum scheduling delay.

Fig. 7. Simulation results of the fairness and delay performance of GMR^3, as compared to DRFQ and MR^3. Figure (a) dedicates to the fairness evaluation, while (b), (c), and (d) compare the scheduling delay of the three schedulers.

for the entire scheduling round until it is processed, incurring a worst-case delay that is as long as the span of an entire round. GMR^3 avoids this problem by distributing the scheduling opportunities over time, in proportion to the flows' weights.

VI. Concluding Remarks

In this paper, we design a new packet scheduler, called Group Multi-Resource Round Robin (GMR^3), that allows independent flows to have a fair share on multiple middlebox resources. GMR^3 collects flows with similar weights into the same flow group, and makes scheduling decisions in a two-level hierarchy. The inter-group scheduler determines a flow group, from which the intra-group scheduler picks a flow in a round-robin manner. Through this design, GMR^3 eliminates the sorting bottlenecks suffered by existing multi-resource scheduling alternatives such as DRFQ, and is able to handle a large volume of traffic at high speeds. More importantly, we show, both analytically and experimentally, that GMR^3 ensures a constant scheduling delay bound that is inversely proportional to the flow's weight, hence offering predictable delay guarantees for individual flows. To our knowledge, GMR^3 is the first multi-resource fair queueing algorithm that offers near-perfect fairness with a constant scheduling delay bound in $O(1)$ complexity.

References

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

[2] 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.

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

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

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

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

[7] A. Greenhalgh, F. Huici, M. Hoerdt, P. Papadimitriou, M. Handley, and L. Mathy, "Flow processing and the rise of commodity network hardware," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 39, no. 2, pp. 20–26, 2009.

[8] 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.

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

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

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

[12] 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.

[13] 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.

[14] 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.

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

[16] 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.

[17] 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.

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

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

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

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

[22] 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.

[23] B. Caprita, J. Nieh, and W. C. Chan, "Group round robin: Improving the fairness and complexity of packet scheduling," in *Proc. ACM ANCS*, 2005.

[24] X. Yuan and Z. Duan, "Fair round-robin: A low-complexity packet scheduler with proportional and worst-case fairness," *IEEE Trans. Comput.*, 2009.

[25] W. Wang, B. Liang, and B. Li, "Multi-resource fair queueing for packet processing with low complexity and bounded delay," University of Toronto, Tech. Rep., 2013. [Online]. Available: http://iqua.ece.toronto.edu/ weiwang/papers/gmr3.pdf

[26] 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.

[27] 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.

[28] S. Y. Cheung and C. S. Pencea, "BSFQ: Bin sort fair queueing," in *Proc. IEEE INFOCOM*, 2002.

[29] C. Guo, "SRR: An $O(1)$ time complexity packet scheduler for flows in multi-service packet networks," in *Proc. ACM SIGCOMM*, 2001.