Low-Complexity Multi-Resource Packet Scheduling for Network Functions Virtualization

Xin Li and Chen Qian
Department of Computer Science, University of Kentucky
Email: xin.li@uky.edu, qian@cs.uky.edu

***Abstract*—Network functions are widely deployed in modern networks, providing various network services ranging from intrusion detection to HTTP caching. Various virtual network function instances can be consolidated into one physical middlebox. Depending on the type of services, packet processing for different flows consumes different hardware resources in the middlebox. Previous solutions of multi-resource packet scheduling suffer from high computational complexity and memory cost for packet buffering and scheduling, especially when the number of flows is large. In this paper, we design a novel low-complexity and space-efficient packet scheduling algorithm called Myopia, which supports multi-resource environments such as network function virtualization. Myopia is developed based upon the fact that most Internet traffic is contributed by a small fraction of elephant flows. Myopia schedules elephant flows with precise control and treats mice flows using FIFO, to achieve simplicity of packet buffering and scheduling. We will demonstrate, via theoretical analysis, prototype implementation, and simulations, that Myopia achieves multi-resource fairness at low cost with short packet delay.**

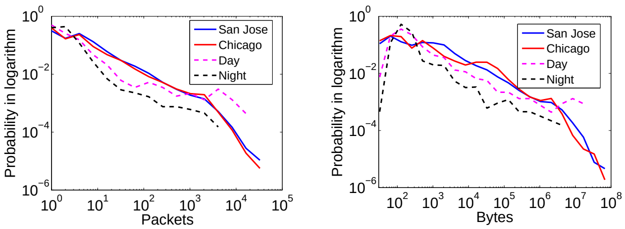
I. Introduction

Network functions (NFs), also known as middleboxes, play an important role in modern computer networks. According to a recent survey [25], NF deployments are ubiquitous: on par with the number of L3 infrastructures. In large-scale enterprise and data centers networks, performance-improving appliances (e.g. WAN optimizers, proxies, and application gateways) become more and more significant, and hence are deployed widely. In addition, security appliances (e.g. firewalls, IDS/IPS) are common in various networks.

On the other hand, the deployment of hardware-based NFs incurs high expenses for physical equipments. When networks continue to grow in both scale and variety, hardware update and replacement also raise big concerns in both laboring and financial cost. To address these challenges, Virtual Network Functions (VNFs), also known as software-centric middleboxes, have been built for general-purpose computing platforms [4] [24]. A wide variety of VNFs [16] could be incorporated into one software middlebox, ranging from intrusion detection to HTTP caching, with high variance of hardware-resource requirements. For example, the resource bottleneck of the intrusion detection task is usually the CPU, but the basic forwarding task is bottlenecked at the network card. In this paper, we focus our discussion on VNF processing in a software middlebox.

Flow isolation is desired in middleboxes to provide predictable services, which means a flow is guaranteed to receive its share of service regardless of the demand of other flows. Flow isolation has been studied extensively, and various packet scheduling algorithms have been proposed to fairly share bandwidth, such as WFQ [9], GPS [20], SFQ [15], and VC [36]. However, these algorithms cannot be applied to VNFs directly, because packet processing in a middlebox needs multiple types of resources, such as CPU, NIC card, and GPU [26]. Ghodsi *et al.* [12] proposed a new multi-resource fair share policy, called Dominant Resource Fairness Queueing (DRFQ) to schedule packets in software middleboxes. Though providing flow isolation, DRFQ incurs high computational complexity and memory space cost. First, DRFQ scheduling algorithm needs to find the flow with the minimum timestamp, whose time complexity is $O(log\, n)$, where n is the number of backlogged flows. Second, each flow should maintain a per-flow buffer to store arriving packets. When the number of flows is large, classification of newly arrived packets and buffering also bring non-trivial time and memory cost. More importantly, the number of per-flow buffers grows linearly with the number of flows. Under the current implementation of fix-sized buffer allocation [12], a big number of buffers waste a large fraction of allocated memory. As a result, the performance of NF processing will be degraded with the increasing number of flows. MR^3 [29] and GMR^3 [30] are proposed to reduce the scheduling time, but the remaining problems have not been resolved.

In this paper, we develop a space-efficient and low-complexity packet scheduling algorithm called *Myopia*, which supports multi-resource environments such as VNFs. Myopia is developed based upon the fact that Internet flow sizes follow power law distribution [32] [17] [23], which indicates that most traffic is contributed by a small fraction of *elephant flows*. We analyze four recent and typical Internet traffic traces and validate the power law distribution of flow sizes. Myopia achieves resource fairness by maintaining the flow state and buffers of elephant flows and scheduling them with precise control. Myopia schedules the majority *mice flows* using the simple FIFO policy, because their limited traffic will not affect the overall fairness. We resolve two main challenges in Myopia: identifying elephant flows and coordinating the scheduling of elephant and mice flows. Our theoretical analysis shows the resource sharing fairness among elephant and mice flows. We implement a Myopia prototype on the Click modular router [18] and evaluate its real processing speed. We also compare Myopia with DRFQ [12] and MR^3 [29] using both



(a) in packet numbers (b) in bytes

Fig. 1. Flow size distribution of the Internet traffic (in logarithm)

real implementation on Click and simulations on NS3, to demonstrate the advantages of Myopia in processing time, fairness, and throughput.

The rest of this paper is organized as follows. We introduce the background of flow size distribution and multi-resource fair sharing algorithms in Sec. II. We give the details of the Myopia design in Sec. III and theoretically analyze its fairness in Sec. IV. We conduct experiments via prototyping on Click and event-driven simulations on NS3 in Sec. V and Sec. VI, respectively. Finally, Sec. VII concludes this paper.

II. Background

A. Power law distribution of flow sizes

Many "power laws" have been discovered in network phenomena, including Internet flow sizes [32] [17] [23], packet inter-arrival time [21], web traffic [7], and the node outdegrees and neighborhood sizes in Internet topologies [11]. Myopia is developed based on the fact that most traffic is generated by a small fraction of flows. Although the power low distribution of Internet flow sizes has been reported by many existing works such as [32] [17] [23], we still provide the analysis of recent Internet traffic data to validate this observation.

The probability density function (pdf) of a power law distribution could be represented mathematically as follows:

$$P[X=x] = \begin{cases} Cx^{-\alpha} & , \quad x \geq x_{min} \\ 0 & , \quad x < x_{min} \end{cases} \qquad (1)$$

where α is called the power law *exponent*. According to the normalization of probability, the integration of Eq. (1) equals 1. Consequently, the constant C is fixed once α is determined: $C = (\alpha-1)x_{min}^{\alpha-1}$.

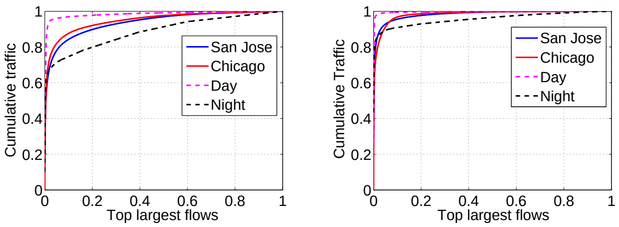
Taking the logarithm operation at the both sides of Eq. (1):

$$ln(P[X=x]) = -\alpha ln(x) + ln(C) \qquad (2)$$

Eq. (2) is used to detect power law distribution: it appears as a straight line when the pdf and x are plotted in logarithmic scale.

We analyze four sets of real Internet traffic traces to illustrate the flow size distribution. We find that power law distribution fits these traces very well.

The first two sets of data are anonymized passive traffic traces from equinix-chicago and equinix-sanjose monitors on high-speed Internet backbone links in 2013, provided by CAIDA [1], denoted as *San Jose* and *Chicago* respectively in this paper. The other two sets of data were captured in the campus network of a U.S. public university at one day in early June of 2014. Both traces were captured for 4 hours, one in the day time and one in the night. The two traces are denoted as *Day* and *Night*, respectively.



(a) in packet numbers (b) in bytes

Fig. 2. Cumulative distribution of Internet flow sizes

α±σ	in packet num	in bytes
San Jose	0.9187±0.0016	0.9398±0.0002
Chicago	0.9182±0.0015	0.9400±0.0003
Day	0.8556±0.0153	0.9107±0.0036
Night	0.8709±0.0143	0.9177±0.0037

TABLE I
THE EXPONENT α AND ERROR σ OF THE TRACES

Fig. 1 demonstrates the flow size distribution of these four traces. We measure these traces from two perspectives: in packet numbers, as depicted in Fig. 1(a), and in bytes, as shown in Fig. 1(b). All curves in Fig. 1 can be approximately fitted as straight lines. Therefore the flow sizes of these four traces can be modeled in power law distribution.

The parameter α of a power law can be calculated via maximum likelihood estimation [19]. [1]

$$\alpha = 1 + n[\sum_{i=1}^{n} ln\frac{x_i}{x_{min}}]^{-1} \qquad (3)$$

The corresponding statistical error σ of Eq. (3) is:

$$\sigma = \sqrt{n}[\sum_{i=1}^{n} ln\frac{x_i}{x_{min}}]^{-1} = \frac{\alpha-1}{\sqrt{n}} \qquad (4)$$

We calculate the values of α and σ of the four traces using maximum likelihood estimation and list them in TABLE I. We can find that more skewed traffic corresponds to smaller value of α. In all four traces, the values of α are very close and the values of σ are very small. We believe these four traces can be considered as representative Internet traffic data, considering both WAN and LAN, both day and night.

Fig. 2 further illustrates the heavy-tail nature of the flow size distributions in the four traces. More than 80% of the total packets are generated by the top 20% largest flows in all four traces. In the Day trace, more than 90% of the total packets are generated by the top 10% largest flows in all four traces. In all four traces, more than 90% of the total traffic in bytes is generated by the top 10% largest flows.

In summary, we demonstrate that today's Internet traffic still approximately follows power law distribution and a small fraction of flows contribute to most traffic. We use *elephant flows* to denote the flows that contribute to most of the traffic and *mice flows* to denote the remaining ones. Following

[1] Extracting α directly from the tangent in the logarithmic plot using least square fitting would introduce systematic error [14].

Algorithm	Time Complexity	Space Complexity
Linear Search	$O(N)$	$O(N)$
Hierarchy Tries	$O(W^d)$	$O(NdW)$
Tuple Space Search	$O(N)$	$O(N)$
TCAM (Hardware)	$O(1)$	$O(N)$

TABLE II
SOME CLASSIC CLASSIFICATION ALGORITHMS ON A CLASSIFIER WITH N FLOWS AND d W-BIT WIDE DIMENSIONS.

[8], elephant and mice flows can be determined by a given threshold, which is further discussed in Sec. III-A.

B. Multi-Resource Scheduling and Current Problems

Network Function Virtualization (NFV) is a recently developed technique that are widely used in ISP networks [27], data center networks [3], [5], [34], and wide-area networks [22]. Dominant Resource Fairness (DRF) was proposed by Ghodsi et al. to generalize max-min fairness to multiple resources in space domain [13]. DRF computes the share of each resource allocated to one user. The maximum share among all resources is called the user's *dominant resource*. Under DRF, the users receive dominant resource in max-min fair fashion. DRFQ is the extension of DRF in the time domain [12]. For a packet p, the dominant resource refers to the resource that p requires most processing time. Motivated by SFQ [15], DRFQ uses virtual time to mimic each flow's resource usages. For DRFQ, one per-flow buffer with a fixed capacity is maintained for each flow. A newly arrived packet will be classified first, and then be stored at the corresponding per-flow buffer. In the meantime, a timestamp is attached to the packet. For each processing opportunity, DRFQ dequeues the packet with the smallest timestamp among those at the head of each per-flow buffer.

As reported by recent work [31] [30], the volume of traffic and number of flows through middleboxes keep increasing. Although providing fair sharing for multi-resource settings, DRFQ (and its variants) still suffers from several efficiency problems when facing a large number of flows.

Classification. A classifier is desired to match a newly arrived packet to its per-flow buffer. Table II lists the spatial and computational complexity of some well-known classification techniques. For all of these techniques, the growing number of flows could impose a challenge for the space and time. Since fast memory such as SRAM and TCAM is expensive and power-hungry, the memory capacity is very limited. People need to either pay more for the hardware or use relatively slow memory to implement the classifier.

Buffering. Fix-sized per-flow buffers are preferred over dynamic buffers, for the following reasons: (1) fix-sized buffers are simpler and faster to access compared to a linked chain of dynamic buffers. Therefore most current implementations of buffer allocation use fix-sized buffering [12] [18]; (2) dynamic buffer allocation requires prediction of flow sizes, which may incur extra complexity and unfairness; (3) each dynamic buffer needs a pointer field, which introduces extra overhead. Worse still, in the fair sharing environment, the longest queue needs to drop packets when the memory space run out [28]. When the number of flows are large, finding longest queue is expensive. In fact fix-sized per-flow buffering is used in DRFQ. We know that sizes of flows vary significantly according to the analysis in II-A. Therefore allocated memory space could be wasted a lot because most flows are mice flows.

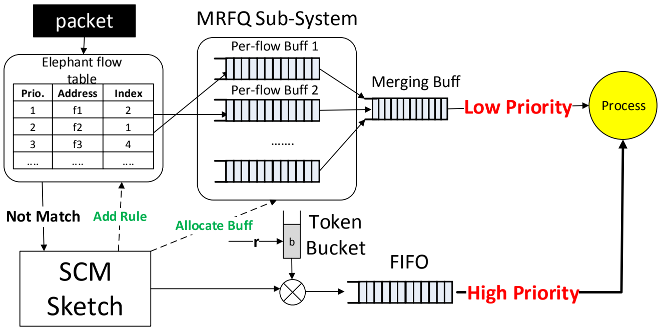


Fig. 3. The framework of Myopia

Scheduling complexity. It is clear that O($logn$) time is needed to find the smallest timestamp before dequeue a packet, where n is the number of backlogged flows. Alternative algorithms, such as MR^3 [29] and GMR^3 [30] try to use round-robin to schedule packets in O(1) time. Nevertheless, their performance may be bottlenecked at the stage of classification and buffering because round-robin also requires per-packet classification and per-flow buffering.

III. MYOPIA DESIGN

The basic idea of our Myopia design is to avoid per-packet classification and per-flow buffering, and hence avoiding the scalability problems in time and space cost as analyzed in Sec. II-B. Myopia takes the advantage of the fact that most of the traffic in a network is contributed by only a very small fraction of flows, which we have demonstrated in Sec. II-A. In Myopia, we propose to precisely schedule the packets of elephant flows to achieve dominant resource fairness among most packets, while let mice flows be scheduled in FIFO. In this way, the resources in a middlebox are still shared fairly while the space and time cost are significantly reduced. Although the idea is straightforward, there are still a number of challenges of the Myopia design, such as identifying the elephant flows and coordinating the scheduling of elephant and mice flows.

The framework of Myopia is shown in Fig. 3. Myopia stores all identified elephant flows in a table using fast memory such as TCAM. It uses a count-min sketch [6] to estimate and size of each flow and identify elephant flows. Packets belonging to an elephant flow will be queued in the per-flow buffer of the flow. Elephant flows are scheduled to satisfy dominant resource fairness. Mice flows are put into a single FIFO buffer. We will explain the details of the Myopia design in the remaining parts of this section.

A. Detecting Elephant Flows

In Myopia, we utilize the count-min sketch (CM Sketch) [6] as the tool to detect elephant flows, for its provable tradeoff between space and accuracy of flow size estimation. We further

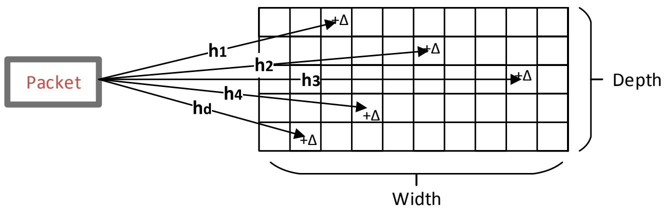


Fig. 4. The CM Sketch update illustration

extend the current CM Sketch design to a shielded count-min sketch (SCM Sketch) that results in less over-estimation.

The CM Sketch is a compact probabilistic data structure to summarize a streamed data, implemented as a two-dimensional array. A CM Sketch is parameterized by two constants, ε and δ, which determine the probabilistic error and space cost of the CM Sketch. With given ε and δ, the width of the CM Sketch w and the depth of the CM Sketch d can both be determined as $w = \lceil \frac{e}{\varepsilon} \rceil$ and $d = \lceil ln\frac{1}{\delta} \rceil$. Also, d pairwise-independent hash functions h_i, $i = 1, 2, ..., d$, are associated with each row in the array. In Myopia, a packet is hashed using its flow ID ID_f, which is usually defined as the 5-tuple in its IP packet header, i.e., $<SrcIP, SrcPort, DstIP, DstPort, Prtcl>$.

$$h_i : \{ID_f\} \rightarrow \{1, 2, ..., w\}, i = 1, 2, ..., d \quad (5)$$

Initially, all entries in the array are 0s. When a new packet p comes, as shown in Fig. 4, the CM sketch update its count as follows for $\forall i, 1 \le i \le d$:

$$count[i][h_i(p.flowID)] \leftarrow count[i][h_i(p.flowID)] + \Delta \quad (6)$$

where Δ is the traffic amount of this packet in Myopia. If the network traffic is measured in packet numbers, $\Delta = 1$. If the network traffic is measured in bytes, Δ is the packet size in bytes.

One can query the CM Sketch for the current size of a given flow f. The query response is calculated as

$$\hat{q}_f = \min_{1 \le i \le d} count[i][h_i(f))] \quad (7)$$

If the size of a flow exceeds a threshold, we identify the flow as an elephant flow.

Like other probabilistic data structures such as the Bloom filter, a CM Sketch only provides approximate results. Over-estimation may occur. The most important feature of the CM Sketch is that there is a provable tradeoff between space cost and accuracy [6]. The estimated query of flow f, $\hat{q}_f$, has the following guarantee with the probability at least $1 - \delta$:

$$\hat{q}_f \le q_f + \varepsilon \sum_{f \in F} q_f \quad (8)$$

where F denotes all flows the middlebox encounters and q_f denotes the exact flow size. $\sum_{f \in F} q_f$ could be interpreted as the total traffic passing through the middlebox. From Eq. (8), the query error of the CM Sketch will monotonically increase over time when the traffic is steadily coming. Thus in Myopia arrays of the CM Sketch should be reset periodically. As a result, the long-live flows (e.g. heartbeats) with few packets would never be recognized as elephant flows. In addition, we propose two techniques to improve the accuracy of the CM Sketch using the same memory space cost, namely *shielding* and *conservative update*. The extended tool is called the shielded count-min sketch (SCM Sketch).

Shielding. When a flow is identified as an elephant flow, there is no need for it to be processed by the SCM Sketch and update the counts in the array. Myopia directly add this flow into a list as shown in Figure 3. As will be described in Sec. III-B, packets of this flow will match the elephant flow table and be directly queued into its per-flow buffer. Packets of identified elephant flows will skip the SCM Sketch step. By shielding elephant flows, the overall over-estimate rate is reduced significantly.

Conservative update. Estan and Varghese introduced the idea of a conservative update [10], which basically means that we endeavour to increment the entries in the array as small as possible. When we need to update, the smallest entry updates as usual; the other entries are set to the maximum of their old value and the new value of the smallest entry. From Eq. (7), we know that it is the smallest entry that matters the query result. The conservative updates reduce their "contribution" to other flows, thus lowering the overall query errors.

In order to determine elephant flows, the threshold T needs to be specified. Every time a new packet arrives at the middlebox, flow size query is conduct immediately after the update process. If the query value is greater than T, the corresponding flow is now recognized as an elephant flow. Another advantage of the SCM Sketch is that $\hat{q}_f \ge q_f$ under all conditions. As a result, all elephant flows would be recognized.

Adding a new component and hash operations may introduce extra time cost. However, it has been reported in [33] that the execution speed of the SCM Sketch using hardware-implemented hash functions [4] [24] can be very fast, which has little affection to packet scheduling and forwarding.

B. Packet scheduling algorithm

For a newly arrived packet p, it will first be checked by the elephant flow table, where each entry is an identified elephant flow. The *index* field in the table contains the pointer to the corresponding per-flow buffer in the MRFQ sub-system. If one entry matches the flow ID of p, p is directly stored in the per-flow buffer of its flow and skip the SCM Sketch step. The deployment of the elephant flow table yields two benefits: (1) as discussed in the previous subsection it shields those elephant flows from the SCM Sketch and lowers the query error, as indicated by Eq. (8); (2) the memory and time cost of checking this table is very low, because elephant flows only consist of a small fraction of all flows.

Packets in the per-flow buffers are scheduled by the multi-resource fair queueing (MRFQ) sub-system. Similar to DRFQ [12], the MRFQ sub-system would use the virtual time to mimic each flow's resource usages. Suppose a packet p is the kth packet of flow i buffered at the per-flow buffer. The notations for the MRFQ sub-system are listed in TABLE III. When packet p arrives, MRFQ computes its virtual start time

Notations	Explanation
p_i^k	k-th packet of flow i
a_i^k	arrival time of packet p_i^k
$s_{i,j}^k$	processing time of p_i^k at resource j
S(p)	virtual start time of packet p
F(p)	virtual finish time of packet p
V(t)	system virtual time at time t
S(p,j)	virtual start time of packet p at resource j
P(t)	the set of packet being processing at time t

TABLE III
NOTATIONS IN THE MRFQ SUB-SYSTEM

and finish time as follows:

$$S(p_i^k) = max\{V(a_i^k), F(p_i^{k-1})\} \tag{9}$$

$$F(p_i^k) = S(p_i^k) + \frac{max_j\{s_{i,j}^k\}}{w_i} \tag{10}$$

The virtual time function is defined as follows:

$$V(t) = \begin{cases} max_j\{S(p,j) | p \in P(t)\} & , \quad P(t) \neq \emptyset \\ 0 & , \quad P(t) = \emptyset \end{cases} \tag{11}$$

When MRFQ dequeues a packet, it chooses one packet of the smallest virtual start time among the packets at the heads of all per-flow buffers.

If a packet p of flow f does not match any entry in the elephant flow table, p will be processed by the SCM Sketch and to check whether it belongs to an elephant flow. Once a new elephant flow is identified, one available per-flow buffer would be allocated for f and a new entry about f will be installed in the elephant flow table. As expected, all incoming packets of f will be queued into f's per-flow buffer.

If p does not belong to any elephant flow, it will be put into the FIFO buffer shared by all mice flows. We choose FIFO here for mice flows because of its simplicity. Since the traffic goes to the FIFO buffer is relatively small compared to the traffic in per-flow buffers, it will not consume too much resource of the middlebox. Suppose that the flow size is subject to power law distribution with exponent α and the CM Sketch threshold T. If $\alpha > 2$, then the expected fraction D of traffic going to the MRFQ sub-system is:

$$D = \frac{\int_T^\infty (x-T)P[X=x]dx}{\int_{x_{min}}^\infty xP[X=x]dx} = \frac{1}{\alpha}\left(\frac{T}{x_{min}}\right)^{-\alpha+2} \tag{12}$$

On the other hand, we define $P(X > T)$, the expected fraction of elephant flows, as P.

$$P = \int_T^\infty P[X=x]dx = (\frac{T}{x_{min}})^{-\alpha+1} \tag{13}$$

By eliminating $\frac{T}{x_{min}}$ in Eq. (12) and Eq. (13), we can get:

$$D = \frac{1}{\alpha} \times P^{(\alpha-2)/(\alpha-1)} \tag{14}$$

By tuning the CM Sketch threshold T, Myopia can determine how much traffic goes to the FIFO buffer and how many flows should be considered as elephant flows. The expected fraction of traffic goes to the FIFO buffer is $1-D$.

In Myopia, all packets in the FIFO buffer have higher priority than those in the MRFQ sub-system. The most important reason for us to choose this design is that it prevents out-of-order packet delivery. For any elephant flow, before being identified by the SCM Sketech, all its packets are queued into the FIFO buffer first. It is possible that when a flow is identified as an elephant flow, some of its packets are still waiting in the FIFO buffer. Emptying the FIFO buffer before starting to schedule packets in the MRFQ sub-system guarantees the intra-flow order. Also, mice flows consume much less resources compared to elephant flows, hence giving them higher priority does not affect the overall fairness. More importantly, preferentially serving the packets from the FIFO buffer actually increases the overall throughput. Gong *et al.* in [17] describe the background and motivation of the preference to mice flows. Most of the mice flows use TCP for transferring packets [17]. If these mice flows are served preferentially, the congestion windows would increase dramatically. Therefore, the throughput also increases. Our simulation using NS3 [2] in Sec. VI-D will show that prioritizing the FIFO buffer does not affect the fairness and improves the overall throughput.

In case the flow sizes do not follow power law distribution, to prevent starvation of the MRFQ sub-system, a token bucket for dominant resource usage is applied to the FIFO buffer to shape its traffic. Upon arriving new packet, if there is no enough tokens, the packet is discarded. Nevertheless, the starvation is not likely to occur, since the fraction of traffic going through the FIFO buffer is really small. The token bucket has two parameters: the token bucket capacity b, and the token rate r. The token bucket capacity b determines the burst size of the FIFO buffer, and the token rate r will upper bound the long term resource utilization of the packets from the FIFO buffer. For example, if $r = 0.3$, the packets in the FIFO buffer should not occupy the hardware resources for more than 30% of the total time in average, thus the MRFQ sub-system will not be starved. How the token rate r adaptively changes according to the number of elephant and mice flows will be our future work. We provide theoretical analysis in Sec. IV about the token bucket algorithm, especially Theorem 5 and Theorem 6. In Sec. VI-C, experiments are conducted to indicate the impact of r on the resource allocation between the MRFQ sub-system and the FIFO buffer.

A merged buffer is used to store the packets with the smallest timestamp from MRFQ sub-system.

Separated token buckets for different resources are also used when we decide when to dequeue a packet to process. Certain number of tokens are taken away from each type of resource based on the packet's processing time on that resource. These token buckets can guarantee those resources are not oversaturated.

C. Improvements of Myopia

Separating detecting and buffering. As we have described in Sec. III-A, the CM Sketch needs to be reset periodically. One challenge is that when we clear the array, there is a great chance that not all per-flow buffers are empty. In

Myopia, we separate the elephant flow *identification phase* and *packet buffering phase*, even though these two are highly related. Only the SCM Sketch array is reset periodically. The classification rules in the elephant flow table and the per-flow buffer allocation remain the same. When the array is reset, packets of elephant flows will skip the SCM Sketch, but the SCM Sketch array will still be updated in the meantime, until it is recognized as elephant flow again.

Elephant flow deallocation The number of per-flow buffers is limited, so is the elephant flow table size. Without a refreshing mechanism, the space of the per-flow buffers and the elephant flow table will soon run out. While it is possible to check FIN packets of TCP flows to determine the end of the flow, it cannot be applied to UDP flows. One simple method is used in Myopia to refresh elephant flows to accommodate newly arrived ones. If a per-flow buffer becomes empty and no packet is being processed, then this buffer will be deallocated and the corresponding classification rule in the elephant flow table will also be deleted. It is based on one observation: the flow size of the elephant flow is also limited, and its packet inter-arriving time is subject to the power law distribution [21]. An empty per-flow buffer usually indicates the end of the flow. People may ask what if we deallocate an active flow. Actually, Myopia can handle such situation very gracefully: when a packet from that flow comes, Myopia can immediately identify the flow as an elephant flow from the SCM Sketch. Myopia then allocates a new per-flow buffer and installs the classification rule again. We do not want this packet to go through the FIFO buffer. That is why Myopia is designed to continue to update the SCM Sketch after its resetting. Another advantage of this design is that the timestamp of an elephant flow is retained even if it is deallocated. Recalling from Eq. (9), if the virtual time $V(a_i^k)$ is greater than $F(p_i^{k-1})$, the timestamp is simply the virtual arrival time. And it is the case when the per-flow buffer becomes empty.

IV. Theoretical Analysis

A. Fairness among Elephant Flows

Let L_i donate the maximum single-resource processing time of the flow i, that is:

$$L_i = \max_{k,j}\{s_{i,j}^k\} \tag{15}$$

Lemma 1. *For any backlogged elephant flow i at any time t,*

$$0 < S\left(p_i^{h(t)}\right) - S\left(p^{l(t)}\right) < \frac{L_i}{w_i} \tag{16}$$

where $p_i^{h(t)}$ denotes the packet of flow i at the head of its per-flow buffer at time t; $p^{l(t)}$ denotes the last packet that has been scheduled before time t. Let $S(\cdot)$ be the timestamp of a packet.

Proof: We proof Lemma 1 by contradiction. Assume that $S\left(p_i^{h(t)}\right) - S\left(p^{l(t)}\right) \geq \frac{L_i}{w_i}$. According to Eq. (10), $S\left(p_i^{h(t)}\right) - S\left(p_i^{h(t)-1}\right) = \frac{max_j\{s_{i,j}^{h(t)-1}\}}{w_i} < \frac{L_i}{w_i}$. Then we could get $S\left(p^{l(t)}\right) - \left(p_i^{h(t)-1}\right) < 0$, which means that $p_i^{h(t)-1}$ is still in the per-flow buffer, which contradict that $p_i^{h(t)}$ is at the head of the buffer. ■

Corollary 1.1. *For any two backlogged elephant flows i_1 and i_2, at any time t*

$$\left|S\left(p_{i_1}^{h(t)}\right) - S\left(p_{i_2}^{h(t)}\right)\right| < max\left(\frac{L_{i_1}}{w_{i_1}}, \frac{L_{i_2}}{w_{i_2}}\right) \tag{17}$$

Proof: Without loss of generalization, we assume that $S\left(p_{i_1}^{h(t)}\right) - S\left(p_{i_2}^{h(t)}\right) \geq 0$. $S\left(p_{i_1}^{h(t)}\right) - S\left(p_{i_2}^{h(t)}\right) = \left[S\left(p_{i_1}^{h(t)}\right) - S\left(p^{l(t)}\right)\right] - \left[S\left(p_{i_2}^{h(t)}\right) - S\left(p^{l(t)}\right)\right] < \frac{L_{i1}}{w_{i_1}} - 0$. It is likewise when $S\left(p_{i_1}^{h(t)}\right) - S\left(p_{i_2}^{h(t)}\right) \leq 0$, thus we can arrive at Corollary 1.1. ■

If the dominant resource of a flow does not change over time, we call that flow *monotonic flow*. Let $W_i(t_1,t_2)$ donate the total processing time consumed by flow i on its dominant resource during $[t_1,t_2)$.

Lemma 2. *For any backlogged dominant-resource monotonic elephant flow i,*

$$\frac{W_i(t_1,t_2)}{w_i} = S\left(p_i^{h(t_2)}\right) - S\left(p_i^{h(t_1)}\right) \tag{18}$$

Proof: Since elephant flow i is monotonic, for any k, $max_j\{s_{i,j}^k\} = s_{i,d_i}^k$, where d_i denotes the dominate resource of flow i. As a result, $W_i(t_1,t_2) = \sum_{k\in[h(t_1),h(t_2))} s_{i,d_i}^k$. According to Eq. (9) and Eq. (10), $S(p_i^{k+1}) - S(p_i^k) = \frac{s_{i,d_i}^k}{w_i}$. Then $\frac{W_i(t_1,t_2)}{w_i} = \sum_{k\in[h(t_1),h(t_2))} S(p_i^{k+1}) - S(p_i^k) = S(p_i^{h(t_2)}) - S(p_i^{h(t_1)})$. ■

Theorem 3. *For any two backlogged dominant-resource monotonic elephant flows i_1 and i_2,*

$$\left|\frac{W_{i_1}(t_1,t_2)}{w_{i_1}} - \frac{W_{i_2}(t_1,t_2)}{w_{i_2}}\right| < 2\times max\left(\frac{L_{i_1}}{w_{i_1}}, \frac{L_{i_2}}{w_{i_2}}\right) \tag{19}$$

Proof: Theorem 3 would be easily induced by Corollary 1.1 and Lemma 2. $\left|\frac{W_{i_1}(t_1,t_2)}{w_{i_1}} - \frac{W_{i_2}(t_1,t_2)}{w_{i_2}}\right| = \left|\left[S\left(p_{i_1}^{h(t_2)}\right) - S\left(p_{i_2}^{h(t_2)}\right)\right] - \left[S\left(p_{i_1}^{h(t_1)}\right) - S\left(p_{i_2}^{h(t_1)}\right)\right]\right| \leq \left|S\left(p_{i_1}^{h(t_2)}\right) - S\left(p_{i_2}^{h(t_2)}\right)\right| + \left|S\left(p_{i_1}^{h(t_1)}\right) - S\left(p_{i_2}^{h(t_1)}\right)\right| < 2\times max\left(\frac{L_{i_1}}{w_{i_1}}, \frac{L_{i_2}}{w_{i_2}}\right)$ ■

This theorem indicates that all elephant flows fairly share the dominant resource. The dominant resource usage difference for any two elephant flows is bounded.

B. Fairness among Mice Flows

At first, the token bucket capacity b should be big enough to accept all kinds of single packet. That is:

$$b > \max_{f\in F}\{L_f\} \tag{20}$$

Let $R(p,j)$ denote the processing time of packet p at resource j and the dominant resource processing time of packet p is denoted as $p^{\uparrow}$, that is $p^{\uparrow} = \max_j\{R(p,j)\}$. Then we can get Lemma 4.

Lemma 4. *Assume packet p_F^h is the head packet at the FIFO buffer. And another packet p_c is the latest packet to complete processing. The maximum delay to start serving packet p_F^h is bounded by:*

$$D(p_F^h) \leq \max_j \{R(p_c, j)\} = p_c^{\uparrow} \tag{21}$$

Proof: Suppose packet p_c is lastly processed by resource i. After $R(p_c, i)$'s time, the bucket token for each resource is ready, hence packet p_F^h could start to be served. $D(p_F^h) = R(p_c, i) \leq \max_j\{R(p_c, j)\} = p_c^{\uparrow}$. ■

Theorem 5. *The maximum delay to start serving packet p from the FIFO buffer is bound by token bucket capacity b:*

$$D(p) \leq b \tag{22}$$

Proof: Suppose there are n packets in front of packet p in the FIFO buffer, which is denoted as $p_i, i = 1, 2, ..., n$, respectively. $D(p) \leq \sum_{i=1}^n p_i^{\uparrow} \leq b$. ■

Since every flow, elephant or mice, would go through the FIFO buffer first, the startup latency for each flow is also bounded by the token bucket capacity b.

C. Relation between Elephant Flows and Mice Flows

Theorem 6. *Assume token bucket rate is r. Packets from the FIFO buffer will at most occupy r portion of the process time.*

Proof: If the FIFO buffer is backlogged, then all resources are serving the packets from the FIFO buffer. Suppose there is a *dummy packet p_d* after last real packet in the FIFO buffer. The FIFO buffer occupation time is actually the process delay of the p_d. According to Lemma 4, $D(p_d) \leq \sum_{i=1}^n p_i^{\uparrow}$, where p_i denotes the ith packet in the FIFO buffer. For each unit time, $D(p_d) \leq \sum_{i=1}^n p_i^{\uparrow} \leq r$ on average. Thus, the FIFO buffer take at most r's time in each unit time. Theorem 6 is proved. ■

V. Prototype Implementation

We have implemented a prototype of Myopia on the Click modular router [18]. The main purpose of this section is to evaluate Myopia's processing speed and demonstrate its advantage in real hardware. We run Myopia on a quad-core@3.40G desktop with 16GB memory. Our Click version is 2.0.1, running in the user-level mode. In order to focus on the bottleneck in the scheduling process, we configure Click to transmit and receive packets within the same machine via memory. That is to say, the real transmission speed is bounded by the memory bandwidth. For the same reason, we skip the packet's real processing modules and the bucket r is set to infinite. Packets are emulated to be serially processed by CPU first then followed by NIC-card, as a multi-resource setting. The CPU processing time of different services follows a simple linear model, as provided by [29], which is listed in TABLE IV. In this table, Bytes means the packet size in bytes. The NIC-card processing time is proportional to the packet size, and the output bandwidth of the middlebox is set to 500 Mbps in our emulation.

Service	CPU processing time (μs)
Basic Forwarding	0.00286×Bytes+6.2
Statistics Monitoring	0.0008×Bytes+12.1
IPSec	0.015×Bytes+84.5

TABLE IV
CPU processing time for different services

	# of Packets	# of Flows
Synthetic	5,084	1,000
Day	135,661	2,294
Chicago	1,021,724	65,212

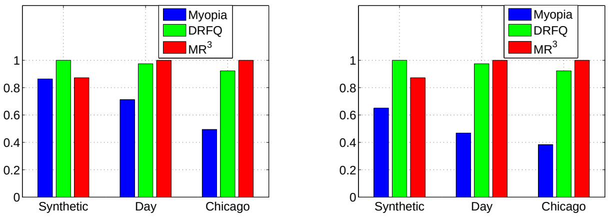
TABLE V
Some statistics of the 3 traces

Our implementation adds two new modules to Click: the SCM Sketch module and MRFQ sub-system module. For the SCM Sketch module, the array is set to 256×4 with MD5 as the hash function. The threshold is set to $13,000$ Bytes.

Three different traces are used for experiments. They are transferred using UDP in the emulation. Each flow is randomly assigned one of the network services in TABLE IV. All flows are transferred simultaneously. We manually generate $1,000$ flows, the size of which is subject to power law, with $\alpha = 2.1$ and $x_{min} = 1,300$ Byte. We call this trace *Synthetic*. The other two traces are Day and Chicago that we have described in Sec. II. Since Chicago is too big to be completely processed, we only use part of it. The statistics of the three traces are shown in Table V. We measure the running time of the above mentioned traces using three different algorithms: Myopia, DRFQ [12] and MR^3 [29]. Fig. 5 illustrates the results, where the running time is normalized. It is obvious that with growing number of flows, the relative speed of Myopia becomes better. One thing needs to be pointed out is that the SCM Sketch calculation time is included the Click emulation, but actually we could implement SCM Sketch in hardware [4] [24] with low cost whose processing time could be neglected [33]. Fig. 5(a) is the raw time comparison, including the overhead from the SCM Sketch. Fig. 5(b) excludes such overhead. In both figures, Myopia has the fastest processing speed. If we consider the factor that Myopia could use SRAM instead of DRAM, the performance of Myopia could be better.

VI. Simulation Evaluation

We also implement Myopia on the NS3 simulator [2]. The basic setup used in this section is the same as what we described in Sec. V. However the traces used may be different in different sets of experiments.



(a) With CM Sketch overhead

(b) Without CM Sketch overhead

Fig. 5. The speed comparison of the three algorithms.

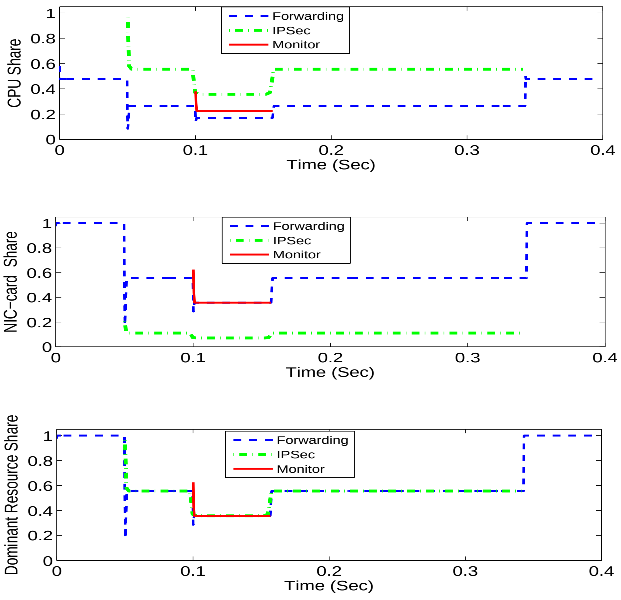
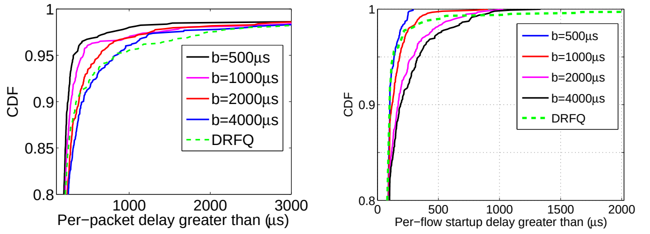


Fig. 6. Dynamics of resource share over time

A. *Flow isolation and fairness*

This set of micro-scale experiments demonstrate that Myopia can quickly respond to newly arrived flows and maintain flow isolation and dominant resource fairness. The trace used here consists of only three UDP flows for three different services: basic forwarding, IPSec and statistical monitoring, respectively. The traffic incoming rates for all three flows are $1,040$ Mbps, all sending 1,300-byte packets. Note that the basic forwarding flow is bottlenecked at the NIC-card, the IPSec flow is CPU-bounded, and the monitoring flow requires most processing time in the NIC-card.

These flows start and end at different time points. Fig. 6 illustrates the dynamics of the resource share over time. Initially, the forwarding flow has 100% share of the NIC-card but only about 50% of the CPU, because its dominant resource is the NIC-card. When the second flow (IPSec) arrives at $5,000$ μs, the forwarding flow experiences a dramatic drop in the resource share and soon roared up to its fair share. On the other hand, the dominant share of the IPSec flow is near 100% at first and soon reached its fair share. The reason is straightforward: Myopia has a preference towards newly coming flows because it will be first placed to the FIFO buffer. When the IPSec flow is identified as an elephant flow, the dominant resource shares of the two flows are equalized. It could be seen that the duration of such glitch is very short, which would not cause serious problems. The join of the monitoring flow has no significant impact to the dominant resource share. Note that DRF is attained throughout this simulation, even when flows come and leave. From the above description, we know Myopia is able to quickly adapt to traffic dynamics and achieve DRF across flows. We also find that Myopia achieves good dominant resource fairness and provides good flow isolation.



(a) CDF of per-packet delay (b) CDF of per-flow startup delay

Fig. 7. CDF of per-packet delay and startup per-flow delay

B. *Scheduling delay*

In this set of experiments, we evaluate two metrics that are widely used in the fair queueing literature to measure the scheduling delay: per-packet delay [12] and per-flow startup delay [29]. The first metric measures the delay from the time when a packet arrives to the time when it finishes service on all resource, and the second one measures how long it takes for a inactive flow to receive service. The distribution of these two delays is also an important indication for fairness. In our simulation, we set the bucket capacity b to $500\mu s$, $1,000\mu s$, $2,000\mu s$ and $4,000\mu s$, respectively. The token bucket rate r is set to as high as 1, so that r will not be the bottleneck. Fig. 7(a) illustrates CDF of the per-packet delay for different token bucket capacities. Myopia achieves shorter per-packet delay in most cases compared to DRFQ. In Fig. 7(b), we compare the per-flow startup delay of Myopia and that of DRFQ. Even though DRFQ has a similar CDF curve as Myopia, it loses in the top 1% percentile and the maximum startup delay for DRFQ ($13,236\mu s$) is much larger (10x) than that of Myopia.

C. *FIFO Buffer V.S. MRFQ Sub-system*

As discussed in Sec. IV-C, the FIFO buffer resource utilization is bounded by the token rate r. The token bucket capacity b is set to $1,000\mu s$. We show the FIFO resource utilization rate in Fig. 8. The peak at the very beginning of both curves is due to the token bucket capacity b to arrow the burst traffic. We can see that a relatively large r does not have too much influence on Myopia. In case the traffic does not follow power law, r is set to prevent starvation of the MRFQ sub-system without the token bucket.

D. *Throughput*

In this section, the token bucket capacity b is set $1000\mu s$ and rate r is set to 0.5.

Throughput of Myopia under UDP flows. A balanced throughput for each flow is another important metric of fairness. Fig. 9 illustrates the throughput of three algorithms: Myopia, DRFQ and FIFO under UDP flows. FIFO is the worst in fairness: a few elephant flows have very high throughput but the other flows are blocked. One the other hand, DRFQ and Myopia have similar performance. Most values of flow throughput fall in the range of 5 MB/s and 60 MB/s. The intra-flow difference is due to their flow-size variance.

Throughput of Myopia under TCP flows. We will see that Myopia will even have a better performance than DRFQ under

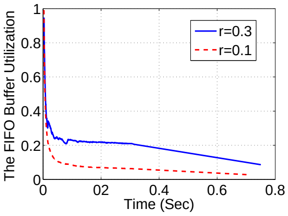


Fig. 8. FIFO utilization

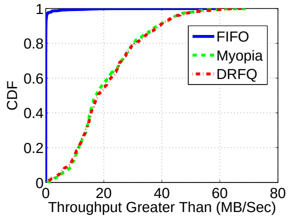


Fig. 9. Per-flow thpt. (UDP)

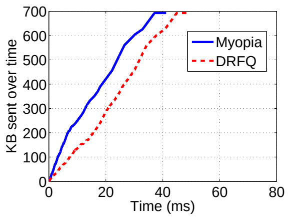


Fig. 10. Bytes sent (TCP)

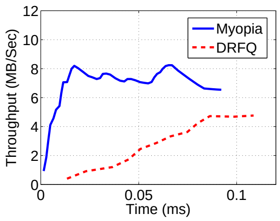


Fig. 11. Large flow thpt. (TCP)

TCP flows. The simulation is done in NS3 [2]. Simultaneously, computer A sends 50 files to another computer B, via the middlebox using TCP. The file size is subject to power law, with $\alpha = 2.1$, $x_{min} = 2{,}000$ Bytes. The overall traffic (in bytes) sent over time is plotted in Fig. 10 for both Myopia and DRFQ. Regarding throughput, we can see that Myopia has a better performance at the very beginning, followed by similar overall throughput in the rest of time. The better performance of Myopia at the beginning is mainly due to the preference to the mice flows. The throughput of large flows over time is shown in Fig. 11. It is clear that Myopia has a higher throughput for the elephant flows at all time compared to DRFQ.

VII. Conclusions

We design a new multi-resource fair scheduling algorithm, called Myopia, which aims at providing dominant resource fairness at low time and space cost by taking the advantage of the fact that Internet flow sizes follow the power law distribution. By separating scheduling of elephant and mice flows, Myopia only needs to maintain very few per-flow buffers and state, thus reducing the computation and memory allocation cost. Theoretical analysis, protoptype implementation, and simulations show that Myopia achieves faster processing time, better fairness, and higher throughput compared to existing solutions. In future we plan to apply multi-resource scheduling to other applications such as traffic measurement [35].

References

[1] The caida ucsd anonymized internet traces 2013 - 2014. mar. http://www.caida.org/data/passive/passive_2013_dataset.xml.

[2] NS3. http://www.nsnam.org/.

[3] M. Al-Fares, A. Loukissas, and A. Vahdat. A Scalable, Commodity Data Center Network Architecture. In *Proc. of ACM SIGCOMM*, 2008.

[4] A. Anand, A. Gupta, A. Akella, S. Seshan, and S. Shenker. Packet caches on routers: the implications of universal redundant traffic elimination. In *Proc. of ACM SIGCOMM*, 2008.

[5] T. Benson, A. Akella, and D. Maltz. Network traffic characteristics of data centers in the wild. In *Proc. of ACM IMC*, 2010.

[6] G. Cormode and S. Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms*, 55(1):58–75, 2005.

[7] M. E. Crovella and A. Bestavros. Self-similarity in World Wide Web traffic: Evidence and possible causes. *Proc. of SIGMETRICS*, 1996.

[8] W. Cui and C. Qian. Difs: Distributed flow scheduling for adaptive routing in hierarchical data center networks. In *Proc. of ACM/IEEE ANCS*, 2014.

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

[10] C. Estan and G. Varghese. New Directions in Traffic Measurement and Accounting. In *Proc. of ACM SIGCOMM*.

[11] M. Faloutsos, P. Faloutsos, and C. Faloutsos. On power-law relationships of the internet topology. In *Proc. of ACM SIGCOMM*, 1999.

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

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

[14] M. L. Goldstein, S. A. Morris, and G. G. Yen. Problems with fitting to the power-law distribution. *The European Physical Journal B-Condensed Matter and Complex Systems*, 41(2):255–58, 2004.

[15] P. Goyal, H. M. Vin, and H. Chen. Start-time fair queueing: a scheduling algorithm for integrated services packet switching networks. In *Proc. of ACM SIGCOMM*, 1996.

[16] R. Guerzoni et al. Network functions virtualisation: an introduction, benefits, enablers, challenges and call for action, introductory white paper. In *SDN and OpenFlow World Congress*, 2012.

[17] L. Guo and I. Matta. The war between mice and elephants. In *Proc. of IEEE ICNP*, 2001.

[18] E. Kohler. *The Click Modular Router*. PhD thesis, Massachusetts Institute of Technology, 2000.

[19] M. E. Newman. Power laws, Pareto distributions and Zipf's law. *Contemporary physics*, 46(5):323–351, 2005.

[20] A. K. Parekh and R. G. Gallager. A generalized processor sharing approach to flow control in integrated services networks: the single-node case. *IEEE/ACM Transactions on Networking*, 1(3):344–357, 1993.

[21] V. Paxson and S. Floyd. Wide area traffic: the failure of Poisson modeling. *IEEE/ACM Transactions on Networking*, 3(3):226–244, 1995.

[22] C. Qian and S. Lam. ROME: Routing On Metropolitan-scale Ethernet . In *Proceedings of IEEE ICNP*, 2012.

[23] I. A. Rai, G. Urvoy-Keller, and E. W. Biersack. Analysis of LAS scheduling for job size distributions with high variance. In *Proc. of ACM SIGMETRICS*.

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

[25] J. Sherry and S. Ratnasamy. A Survey of Enterprise Middlebox Deployments. Technical report, EECS, UC Berkeley, 2012.

[26] R. Smith, N. Goyal, J. Ormont, K. Sankaralingam, and C. Estan. Signature Matching in Network Processing Using SIMD/GPU Architectures. In *Proc. of IEEE ISPASS*, 2009.

[27] N. Spring, R. Mahajan, and D. Wetherall. Measuring ISP topologies with Rocketfuel. 2002.

[28] B. Suter, T. V. Lakshman, D. Stiliadis, and A. Choudhury. Design considerations for supporting TCP with per-flow queueing. In *Proc. of IEEE INFOCOM*, 1998.

[29] W. Wang, B. Li, and B. Liang. Multi-Resource Round Robin: A Low Complexity Packet Scheduler with Dominant Resource Fairness. In *Proc. of IEEE ICNP*, 2013.

[30] W. Wang, B. Liang, and B. Li. Low Complexity Multi-Resource Fair Queueing with Bounded Delay. In *Proc. of IEEE INFOCOM*, 2014.

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

[32] W. Willinger, M. S. Taqqu, R. Sherman, and D. V. Wilson. Self-similarity through high-variability: Statistical analysis of ethernet LAN traffic at the source level. *IEEE/ACM Transactions on Networking*, 1997.

[33] M. Yu, L. Jose, and R. Miao. Software defined traffic measurement with opensketch. In *Proc. of USENIX NSDI*, 2013.

[34] Y. Yu and C. Qian. Space shuffle: A scalable, flexible, and high-bandwidth data center network. In *Proceedings of IEEE ICNP*, 2014.

[35] Y. Yu, C. Qian, and X. Li. Distributed collaborative monitoring in software defined networks. In *Proc. of ACM HotSDN*, 2014.

[36] L. Zhang. Virtual clock: A new traffic control algorithm for packet switching networks. In *Proc. of ACM SIGCOMM*, 1990.