vCRIB: Virtualized Rule Management in the Cloud

Masoud Moshref[†] *Minlan Yu*[†] *Abhishek Sharma*[†*] *Ramesh Govindan*[†]
[†] *University of Southern California* [*] *NEC*

Abstract

Cloud operators increasingly need many fine-grained rules to better control individual network flows for various management tasks. While previous approaches have advocated placing rules either on hypervisors or switches, we argue that future data centers would benefit from leveraging rule processing capabilities at both for better scalability and performance. In this paper, we propose vCRIB, a virtualized Cloud Rule Information Base that allows operators to freely define different management policies without the need to consider underlying resource constraints. The challenge in our approach is the design of a vCRIB manager that automatically partitions and places rules at both hypervisors and switches to achieve a good trade-off between resource usage and performance.

1 Introduction

To improve cloud security, network utilization, application performance, and fairness among tenants, operators need increasingly many *fine-grained rules* (e.g., access control rules, rate limiting rules) to better control individual virtual machines and flows. There can be up to 1M-1B rules for one management task in the cloud (see Section 2). Managing these rules becomes challenging when it involves a large number of servers, switches, applications, tenants, and a variety of network management tasks.

Recently, there has been significant research on supporting *individual* management task by proposing new types of rules and new ways to handle these rules. These solutions are designed either exclusively for hypervisors (e.g., Open vSwitch [4, 3], Seawall for rate limiting [21], CloudPolice for access control [20]) or exclusively for switches (e.g., BigSwitch [1], Hedera for traffic engineering [6], FairCloud [19] for fair sharing). However, rule processing in hypervisors can require dedicated CPU resources and switches have limited TCAM memory, so we do not see either approach being used exclusively as the need for fine-grained rules increases.

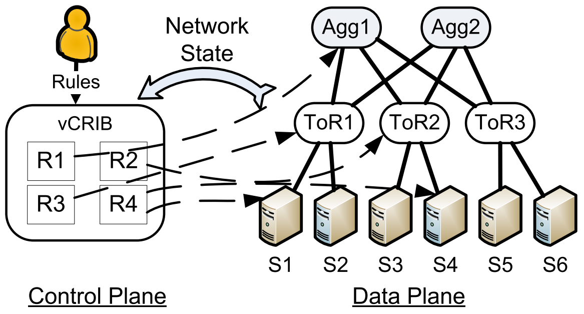


Figure 1: Virtualized Cloud Rule Information Base (vCRIB)

Instead, we argue that future data centers would benefit from a *unified* rule management system across multiple tasks that automatically places rules on switches and hypervisors as necessary and as dictated by the configuration of the data center. The unified rule management system not only allows operators to freely define different management policies without considering underlying resource constraints, but also enables more efficient resource usage by managing different sets of rules together.

Inspired by the virtual memory concept in operating system which allows applications to freely claim and use memory independent of the physical memory size, we propose a *virtualized Cloud Rule Information Base (vCRIB)* that provides operators or a network management system with the abstraction of an unbounded prioritized list of rules (Figure 1). We design a *vCRIB manager* that *jointly* considers the resource, cost, and performance constraints in the hypervisors and switches, and automatically installs rules at the right location to optimize the packet processing performance while minimizing the resource usage and cost. There are three key challenges in designing the *vCRIB manager*: (i) A large number

Policy	# rules	Where
Bw alloc. [21, 19]	O(m)-O(β) [100K-1M]	Hypervisors
Access control [20]	O(β) [$>$ 1M]	Hypervisors
Flow Measurement for TE [6, 12, 8]	O(f) [10M–1B]	Hypervisors or switches
VLAN-based traffic management [18]	O(kn) [$>$1M]	Hypervisors or switches

Table 1: # Rules in the Cloud depending on # VMs (m), #pairs of communicating VMs (β), # active flows (f), # servers (n), and # VLANs (k)

of rules that are defined on different fields (e.g., IP addresses, ports) and may overlap with each other, (ii) limited memory in cheap switches (especially top-of-rack switches), limited CPU resources in hypervisors, and limited bandwidth in data centers, and (iii) some rules may be better handled in hypervisors, while others may be better suited for switches, depending on the rule functions and the traffic that matches the rules. Moreover, the vCRIB manager must also dynamically adjust the representation and placement of the rules in response to the network dynamics such as virtual machine migration and traffic changes.

Our initial approach to the design of the vCRIB manager contains two modules: the partition module separates the placement decisions for overlapping rules by partitioning the rule space and splitting those rules across the partitions; the placement module places partitions of rules instead of individual rules by considering the performance of rule processing, the CPU and memory constraints at hypervisors and switches, and data center topology and routing paths. Our preliminary evaluation of realistic rules shows encouraging results of our partition and placement algorithms.

2 Why Cloud Rule Management is Hard

Increasingly, cloud operators need *fine-grained* rules to better control individual flows. Processing and placement of these rules significantly affect both network performance and resource usage. These two factors make rule management difficult and motivate the need for a virtualized cloud rule information base.

Many Fine-grained Rules. As data centers evolve, operators will specify *high-level policies* for different management tasks such as access control, policy routing, traffic isolation, and QoS. Although there is a wide range of policies, these policies can, in general, be translated into a large number of *low-level rules* that are installed on devices in the data center. We define the *rules* as matching on various packet header fields (e.g., IP addresses, MAC addresses, ports, VLAN tags) and performing various actions on the packets (e.g., drop, rate limit, count)[1].

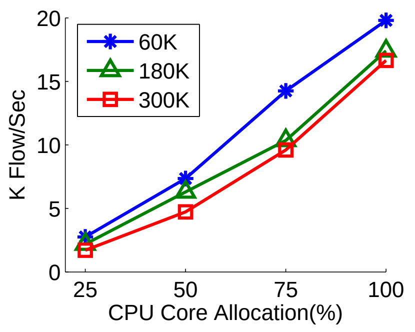
[1]The definition of the rules is a generalization of OpenFlow rules and their extensions [3].

Most policies can be expressed in the general rule format as summarized in Table 1. For example, to ensure each tenant gets a fair share of the bandwidth, Seawall [21] installs rules that match the source VM address and perform rate limiting on the corresponding flows. To enable customized routing for traffic engineering [6, 8] and energy efficiency [12], an operator may need to get traffic statistics using rules that match each flow (defined by five tuples) and count its number of bytes or packets.

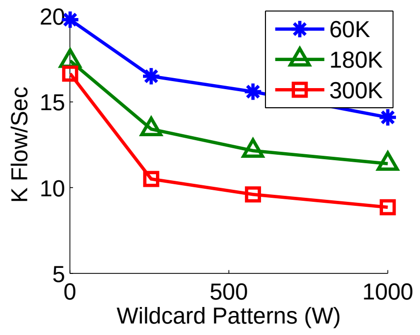
A simple policy can result in a large number of fine-grained rules, especially when it is defined on the number of VMs and servers and the number of active flows. To provide fair bandwidth allocation per source-destination pair [19], we need $\beta = 1M$ rules, when there are 100K VMs where 1% of VMs each communicating with another 1% of VMs. To measure the per flow statistics for traffic engineering, we need 10M to 1B rules in data centers with 10K-100K servers and 1K to 10K active flows per server [14]. For a data center with 10K-100K servers and up to 4K VLANs, we may need up to 400M rules for NetLord [18], which uses multiple VLANs to provide high end-to-end bandwidth for multiple tenants. More examples are summarized in Table 1. In the future, operators will need even more rules for new policies with finer-grained control on individual flows and virtual machines. A combination of these policies would lead to even more rules if we manage them separately. The difficulty of rule management will also be exacerbated by dynamics: as services are reconfigured or as traffic changes, rules may need to be dynamically updated (e.g., the routing rules in [8] change every 1.5 to 5 seconds due to traffic changes).

Hypervisors or Switches? No Silver Bullet. Broadly speaking, two camps have emerged on the question of *where* to place rule processing functionality. One camp, exemplified by some research proposals [21, 20] and industrial solutions [4, 3] advocates placing rules in hypervisors at servers. The other, exemplified by [1, 17, 2], chooses to place rules in the TCAMs at switches inside the network. *We take the position that neither camp is likely to be entirely correct* and future data centers will process rules *both* at the hypervisors and switches.

Software-based hypervisors at servers can support complex operations of the rules (e.g., dynamically calculating rates of each flow [21]). However, processing the rules may require committing an entire core or a substantial fraction of a core at each server in the data center. Data center operators would prefer to allocate as much CPU as possible to client VMs to maximize their revenue. (e.g., RackSpace operators prefer not to dedicate even a portion of a server core for rule processing [5].) We use Open vSwitch, the software switch running in hypervisors, to evaluate the data plane rule processing



(a) Effect of allocated CPU (W=1)



(b) Effect of wildcard rules on an entire core

Figure 2: Open vSwitch data plane performance (Evaluated on a six 2.8 MHz core machine with 6 GB RAM using flows generated by the Open vSwitch benchmark [4].)

performance with different CPU allocation. When we allocate 25% of a core to rule processing, Open vSwitch can only handle 3K new flows/sec with 60K rules (Figure 2(a)). This is because, for each new flow, it should go to the user space to find the match rules. The CPU usage and performance of rule processing also depend on the types of rules. Often, the rules have wildcard patterns (e.g., the IP prefixes with different lengths 10.2.0.0/24 and 10.2.0.0/16 are two wildcard patterns). Figure 2(b) shows that to process 60K rules with 256 wildcard patterns with an entire core, the throughput drops from 20K to 16K when there are more wildcard patterns (W increases from 1 to 256). The reason is that Open vSwitch classifier creates a hashset for each wildcard pattern and goes through them linearly. Therefore, we should carefully consider the allocated CPU for rule processing and the types of rules when we place rules in hypervisors.

In contrast, switches leverage custom silicon to provide more scalable rule processing than hypervisors. However, since TCAMs are expensive and power hungry, switches can support at most thousands of rules [10], and cheaper ToR switches support even fewer.

Moreover, some rule processing functions can be performed more efficiently either in the hypervisor or inside the network, and some are best accomplished by a combination of the two. For instance, placing access control rules in the hypervisor (or at least at the ToR switches) can avoid injecting unwanted traffic into the network. In contrast, operations on the aggregates of traffic (e.g., measuring the traffic traversing the same link) can be easily performed at switches inside the network. Similarly, operations on inbound traffic from the Internet (e.g., load balancing) should be performed at the core/aggregate routers. Rate control is a task that can require cooperation between the hypervisors and the switches. Hypervisors can achieve end-to-end rate control by throttling individual flows or VMs [21], but in-network rate control can directly avoid buffer overflow at switches. Even for data centers where only switches (or hypervisors) have rule processing functionality, vCRIB is still useful, since it can reduce operators' burden by automatically managing the placement of a large number of fine-grain rules.

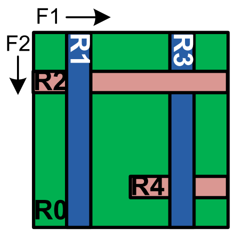
Virtualizing Cloud Rule Management. The need for large numbers of fine-grain rules which support a variety of management tasks, the dynamics of data centers, and the resource costs of rule processing suggest that future data centers will require careful rule management that *partitions* rules and *places* them both in hypervisors and switches to achieve the desired functionality while carefully balancing cost and performance. *Given the complexity of this endeavor, we argue that future data center management systems should modularize rule management into a separable module or service that exports a simple abstraction to the rest of the management system, hiding the details of rule partitioning and placement.*

3 Virtualized Cloud Rule Management

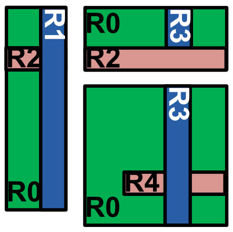
Our cloud rule management system exports a centralized abstraction called vCRIB, *Virtualized Cloud Rule Information Base*. It simply consists of a potentially unbounded list of rules. Each rule describes the flow fields to match and an associated action to be performed on flows matching it. An operator, or other (distributed) network management modules, can insert new rules, delete or re-prioritize existing rules in vCRIB. We design a vCRIB manager to install the rules in hypervisors and switches to handle traffic forwarded from sources based on their CPU and memory resources, the data center topology, and routing information.

Challenges of designing vCRIB manager. The key challenge of designing vCRIB Manager is how to manage and place a large number of overlapping rules with resource and performance concerns.

Many fine-grained rules for different management tasks may overlap with each other. For example, the rate-limiting rules at the source A overlaps with the access



(a) Rules in flow space



(b) Partitions

Figure 3: Partition a flow space

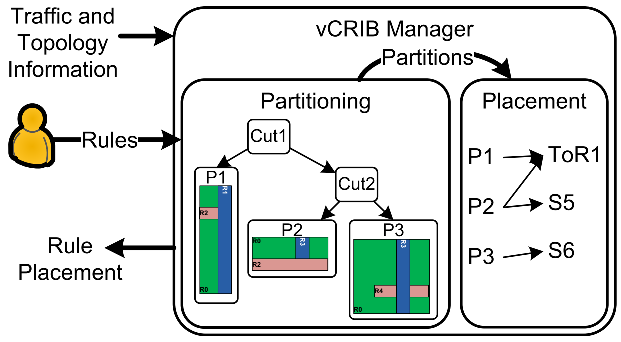


Figure 4: Architecture of vCRIB manager

control rule that blocks traffic to destination B because the packet from source A and destination B matches both rules. In general, the flow space usually has seven or more dimensions (source/destination IP addresses, MAC addresses, ports, the protocol) and each rule is described as actions on a hypercube (a multi-dimensional generalization of a rectangle) in these dimensions. For flows falling in the intersection of two hypercubes, the action corresponding to the higher priority rule must be performed. Figure 3(a) shows a two-dimensional flow space ($F1$, $F2$). There are five rules in the flow space with different actions (as indicated by their colors) and different priorities (e.g., $R1 > R2 > R3 > R4 > R0$).[2]

When rules overlap, we cannot simply place a rule at one switch independent of other rules, since that would result in an incorrect action. For example, in Figure 3(a), since $R3$ has higher priority than $R4$, we cannot place rule $R4$ without placing $R3$. Otherwise, the packets that match both $R3$ and $R4$ will take $R4$'s action instead of $R3$'s. Similarly, we cannot place $R3$ without placing $R2$ and then $R1$. On the other hand, if we relocate $R1$, then $R2$ should be moved to the same location to preserve matching correctness, since $R2$ overlaps with $R1$.

Making placement decisions for many rules and many locations becomes even more challenging when we have performance and resource concerns. For example, memory constraints in a switch or hypervisor may force routing rules to be placed in a way that increases path length, or deny rules to be placed upstream of the sources. These choices cause traffic to traverse over more links than they otherwise might have.

To overcome these challenges, our proposed vCRIB manager has two key modules (Figure 4): the *partition* module to separate the placement decisions for overlapping rules and the *placement* module to consider the resource constraints. In this paper, we focus on the algorithmic challenges and the benefits of managing rules in vCRIB, and defer the question of how to dynamically adapt to vCRIB insertions and deletions to future work.

[2]These rules cover the entire flow space with a default rule (e.g., $R0$) covering the rest of the flow space that the other rules do not cover.

Handling overlapping rules with partitions. A *partition* is a hypercube containing a group of rules, which is the basic unit we use for placement decisions. Rules in a partition should be placed at the same location (a hypervisor or a switch). Multiple partitions may be placed at the same location. To ensure that we can make independent decisions for placing each partition, when a rule breaches a partition boundary, we split the rule into two hypercubes at the partition boundary. Figure 3(b) shows an example of three partitions, where we split $R2$ and $R3$ into two parts each. Within a single partition, however, rules may overlap: for example, $R1$ and the fragment of $R2$ overlap in the first partition.

However, partitioning increases the total number of rules in the flow space because of rule splitting. For example, in Figure 3(b), the total number of rules grows from five to nine after the partition. We define N as the increase of the number of rules ($N = 4$ in the example). The key algorithmic challenge in partitioning is to navigate the trade-offs in finding the right size (number of rules) $S(p)$ for each partition p. More partitions with smaller $S(p)$ increase the number of rules. However, it is easier to place smaller partitions than larger ones on hypervisors and switches in order to satisfy CPU, memory and bandwidth constraints. We address this trade-off by minimizing $F = \alpha \times max_p S(p) + (1-\alpha) \times N$, where α is a configurable parameter. A larger α leads to more flexibility in placing the rules while a smaller α reduces the total number of rules.

In our initial design, we have explored a top-down approach to partition the flow space. We construct a Binary Space Partition (BSP) tree, whose root represents the entire flow space. At each node in the tree, we create two children of nodes as partitions by picking a dimension and dividing the value range in that dimension into two parts. We do this recursively while attempting to minimize the objective function F. The way we pick the dimension and divide each dimension is based on related work in the computational geometry literature [11], which defines an objective function similar to F and proves that the BSP approach can achieve an optimal solution.

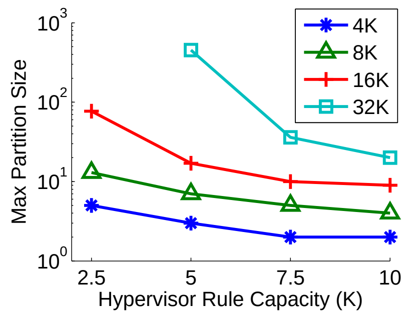
Placing partitions with resource constraints. The placement module of the vCRIB manager must match partitions to locations (i.e., hypervisors and switches) that jointly respect two resource constraints. First, switches may not be able to handle too many rules because of limited TCAM size and servers may incur CPU overhead for rule matching in hypervisors. Second, given these constraints, rule placement may cause flows to be redirected in a way that consumes additional bandwidth resources, and a key algorithmic challenge is to find a placement that minimizes the additional bandwidth resulting from traffic indirection.

Among the many possible placement formulations, we have explored the objective of minimizing the additional bandwidth usage of traffic indirection while constraining the number of rules at different locations.[3] Our formulation can be modeled as the *Generalized Assignment problem*, which is NP-hard. We use a depth-first search (DFS) branch-and-bound approach to search through all the possible mappings between the partitions and the locations. To improve the efficiency of the DFS algorithm, we rank the partitions and locations in specific orders. We sort the partitions in the decreasing order of the number of rules: placing larger partitions first can significantly reduce the search space. For each partition, we identify all the locations that can support the types of rules in the partition. We sort the locations based on the bandwidth overhead: to avoid the extra bandwidth usage, we try to place the rules on the routes of the packets. Moreover, if the action of a rule is to drop the packet, we place the rules as near to the source of the traffic matching the rules as possible.

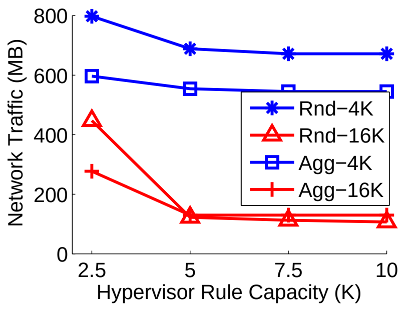
4 Preliminary Evaluation

We find the results from a preliminary evaluation of our partition and placement algorithms with different resource availability scenarios encouraging. Specifically, we model resource constraints on hypervisors by bounding the number of rules allowed on each hypervisor; this indirectly models CPU commitments on each hypervisor for rule processing. We evaluate vCRIB's solution for managing 4K-16K rules on a data center topology (Figure 1) with space for 1K rules on each switch and 2.5K, 5K, 7.5K, and 10K rules on each hypervisor. We

[3]We have deferred other objective functions, and examining partition replication, to future work. We only consider placing a single copy of each partition because short flows have a large share in data centers (e.g., 80% of flows last less than 10s [14]). Hence, we expect the benefits of rule placement at multiple locations to be limited, but have deferred an exploration of this.



(a) Size of partitions



(b) Traffic overhead of placement

Figure 5: Rule partition and placement with different hypervisor capacities

generate the rules and flows using the ClassBench benchmark [22] and the flow sizes as per [14].

With space for more rules in the hypervisors, our partition module chooses smaller partitions (Figure 5(a)). As a result the placement algorithm has more flexibility in placing rules, and it can achieve lower traffic overhead by putting more rules at the source hypervisors (Figure 5(b)).

The efficiency of rule placement also relies on the ways VM IP addresses are assigned. There are two cases: The IP addresses of VMs on the same server are chosen contiguously (*Agg*), which makes it easy to aggregate rules for these VMs as one; and the IP addresses of VMs are not contiguous (*Rnd*) allowing VMs to keep their IP addresses after migration, but the rules for VMs on the same server are not aggregatable. As shown in Figure 5(b), the contiguous address allocation has lower traffic overheads, because aggregating source IPs helps the placement algorithm.

5 Related Work

Data center management systems. As discussed in Section 2, there have been many proposals for *individual* management tasks that manage their specific rules on either hypervisors [4, 9, 21, 20]) or switches [1, 6, 19].

vCRIB is complementary to these systems and provides a single abstraction for all types of rules, decoupling rule definition from rule partitioning and placement. vCRIB frees these systems from the complexity inherent in the rule management and enables better resource usage by considering rules from different tasks together.

Rule partition and placement solutions. The problem of partitioning and placing multi-dimensional data at different locations also appears in software defined networking [23], distributed databases and systems [16, 15], and computational geometry [11]. The key difference is that vCRIB should consider the *interactions* of partition and placement modules in the *data center context* (e.g., hypervisors vs switches, multi-rooted tree topology, many short flows). Compared with DIFANE [23], which *randomly* places *a single* partition of rules at each switch, vCRIB optimizes partitions' placement to reduce bandwidth usage by dividing the flow space into many small partitions and placing *multiple* partitions at each location. The systems in [16, 15] focus on multi-dimensional data, and use R-tree and its extensions to reduce the number of hypercubes after partitioning, but optimize the placement of partitions for load balancing only. In contrast, vCRIB considers network topology and where the packets come from. vCRIB borrows ideas from computational geometry [11], but also considers placement efficiency during partitioning.

Distributed Firewall. Distributed firewalls [7, 13], often used in enterprises, leverage a centralized manager to deploy security policies on edge machines. vCRIB manages more fine-grained rules on individual flows and virtual machines for various policies including firewalls in the cloud. Rather than placing these rules at the edge machines, vCRIB places these rules given the limited rule processing resources at both hypervisors and switches while minimizing traffic overhead.

6 Conclusion and Future Work

vCRIB provides data center operators an abstraction for specifying and managing various rules. vCRIB automatically partitions and places the rules on hypervisors and switches to achieve the best trade-off of performance and cost. For future work, we will study better combinations of partition and placement modules, understand the implementation constraints on network state information accuracy, and design online algorithms to handle network and rule dynamics.

References

[1] Big Switch Networks. `http://www.bigswitch.com/`.

[2] NEC's OpenFlow Networking solution for Genesis. `http://www.nec.com/en/case/genesis/index.html`.

[3] Nicira whitepaper: It's Time to Virtualize the Network. `http://nicira.com/en/network-virtualization-platform`.

[4] Open vSwitch. `http://openvswitch.org/`.

[5] Private conversation with rackspace operators.

[6] AL-FARES, M., RADHAKRISHNAN, S., RAGHAVAN, B., HUANG, N., AND VAHDAT, A. Hedera: Dynamic flow scheduling for data center networks. In *NSDI* (2010).

[7] BELLOVIN, S. M. Distributed firewalls. *;login:* (1999).

[8] BENSON, T., ANAND, A., AKELLA, A., AND ZHANG, M. MicroTE: Fine Grained Traffic Engineering for Data Centers. In *ACM CoNEXT* (2011).

[9] CASADO, M., KOPONEN, T., RAMANATHAN, R., AND SHENKER, S. Virtualizing the network forwarding plane. In *PRESTO* (2010).

[10] CURTIS, A., MOGUL, J., TOURRILHES, J., YALAGANDULA, P., SHARMA, P., AND BANERJEE, S. Devoflow: Scaling flow management for high-performance networks. In *SIGCOMM* (2011).

[11] D'AMORE, F., NGUYEN, V. H., ROOS, T., AND WIDMAYER, P. On optimal cuts of hyperrectangles. *Computing* (1995).

[12] HELLER, B., SEETHARAMAN, S., MAHADEVAN, P., YIAKOUMIS, Y., SHARMA, P., BANNERJEE, S., AND MCKEOWN, N. ElasticTree: Saving Energy in Data Center Networks. In *NSDI* (2010).

[13] IOANNIDIS, S., KEROMYTIS, A. D., BELLOVIN, S. M., AND SMITH, J. M. Implementing a distributed firewall. In *CCS '00* (2000).

[14] KANDULA, S., SENGUPTA, S., GREENBERG, A., PATEL, P., AND CHAIKEN, R. The nature of datacenter traffic: Measurements and analysis. In *IMC* (2009).

[15] KRIAKOV, V., DELIS, A., AND KOLLIOS, G. Management of highly dynamic multidimensional data in a cluster of workstations. *Advances in Database Technology-EDBT 2004* (2004).

[16] MONDAL, A., KITSUREGAWA, M., OOI, B. C., AND TAN, K. L. R-tree-based data migration and self-tuning strategies in shared-nothing spatial databases. In *GIS* (2001).

[17] MUDIGONDA, J., YALAGANDULA, P., AL-FARES, M., AND MOGUL, J. SPAIN: COTS Data-Center Ethernet for Multipathing over Arbitrary Topologies. In *NSDI* (2010).

[18] MUDIGONDA, J., YALAGANDULA, P., MOGUL, J., AND STIEKES, B. NetLord: A Scalable Multi-Tenant Network Architecture for Virtualized Datacenters. In *SIGCOMM* (2011).

[19] POPA, L., KRISHNAMURTHY, A., RATNASAMY, S., AND STOICA, I. Faircloud: sharing the network in cloud computing. In *HotNets* (2011).

[20] POPA, L., YU, M., KO, S. Y., STOICA, I., AND RATNASAMY, S. CloudPolice: Taking access control out of the network. In *HotNets* (2010).

[21] SHIEH, A., KANDULA, S., GREENBERG, A., KIM, C., AND SAHA, B. Sharing the datacenter networks. In *NSDI* (2011).

[22] TAYLOR, D. E., AND TURNER, J. S. ClassBench: a packet classification benchmark. *IEEE/ACM Transactions on Networking* (2007).

[23] YU, M., REXFORD, J., FREEDMAN, M. J., AND WANG, J. Scalable flow-based networking with difane. *SIGCOMM* (2010).