SideCar: Building Programmable Datacenter Networks without Programmable Switches

Alan Shieh[‡†] Srikanth Kandula[‡] Emin Gun Sirer[†]
[‡] Microsoft Research and [†] Cornell University

Abstract— This paper examines an extreme point in the design space of programmable switches and network policy enforcement. Rather than relying on extensive changes to switches to provide more programmability, SideCar distributes custom processing code between shims running on every end host and general purpose *sidecar* processors, such as server blades, connected to each switch via commonly available redirection mechanisms. This provides applications with pervasive network instrumentation and programmability on the forwarding plane. While not a perfect replacement for programmable switches, this solves several pressing problems while requiring little or no change to existing switches. In particular, in the context of public cloud data centers with 1000s of tenants, we present novel solutions for multicast, controllable network bandwidth allocation (e.g., use-what-you-pay-for), and reachability isolation (e.g., a tenant's VM only sees other VMs of the tenant and shared services).

Categories and Subject Descriptors

C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design

General Terms

Algorithms, Design, Performance, Security

Keywords

Datacenters, programmable switches, virtualized networks.

1. INTRODUCTION

The case for switches that are more programmable has long been made. Programmable switches can be used for better monitoring [32], improved security enforcement [7], explicit feedback for more adept congestion control [29], and other novel features. However, such switches remain a work-in-progress [26, 33, 34].

This paper examines an extreme point in the design space– a network wherein switches are augmented with an external general purpose *sidecar* processor, but are otherwise minimally modified, i.e., no internal changes to the software or hardware of the switch. With these constraints, SideCar enables applications to install custom packet processing rules that execute within the network; these rules consist of a packet classifier, combined with associated code that processes every packet matching that classifier.

Our key insight in realizing this programming model entails pushing packet classification to the edge and offloading custom processing to commodity servers. By having end hosts designate packets as needing special processing and having switches redirect designated packets, the hardware requirements for switches are substantially reduced: each switch need only process a small set of packet classifiers rather than a large set of complex packet formats. By limiting modifications to already-open end host platforms and treating traditionally-closed switch platforms as black boxes, SideCar is compatible with entrenched industry practices.

SideCar relies on packet sampling at switches to improve scalability: it reduces the volume of traffic that needs to be inspected. Sampling also enables defense in depth: relying on edge marking to implement fundamental network safety properties expands the trusted computing base, rendering these safety properties vulnerable to compromised end hosts. SideCar prevents these by random spot checking and, upon detection, punishment of misbehavior.

The SideCar execution model is inspired by the slow-path packet processing used by all modern switches. To achieve cost and performance requirements, the majority of traffic on a switch is processed by a comparatively inflexible datapath. While many applications and protocols can benefit from richer processing of only a small fraction of their traffic, the control processors in today's switches are closed, hard to program and strapped for resources. With SideCar, switches process such special packets and events by redirecting them to a sidecar processor.

Two trends in software and network architecture enable sidecars. First, connecting a commodity server with a 1 Gbps NIC to top-of-rack switches suffices for surprisingly many applications. By exploiting GPUs, multi-core CPUs and multi-queue NICs for parallel processing and high bandwidth point-to-point interconnects in lieu of a shared bus, recent efforts [13, 19] show that one server can process packets at up to 40Gbps.

[†]{ashieh,egs}@cs.cornell.edu Dept. of Computer Science, Cornell University, Ithaca, NY.
[‡]srikanth@microsoft.com Microsoft Research, Redmond, WA.

Hotnets '10, October 20–21, 2010, Monterey, CA, USA.

Second, today's switches and the data center edge provide many alternatives to sample or steer packets to sidecars. Many commodity switches implement sFlow, which can copy a sampled subset of packets arriving at one or more ports onto another port. Switches can also steer packets based on TCAM filters or VLAN tags. The datacenter edge can mark packets and implement functionality in conjunction with sidecars. In private datacenters that are managed by automatic software (e.g., AutoPilot [21]), shim layers can be added to the OS network stack. In cloud datacenters that offer virtualized resources for pay (e.g., Azure, AWS), the hypervisor (or the root partition or dom-0) can mark packets for inspection by sidecars. For both types of datacenters, instrumentation at sidecars can be used to provide precise feedback to better configure packet filters and rate limiters at the edge.

We present an initial design of the SideCar architecture. By building a small set of applications with SideCar, we explore the challenges in choosing appropriate steering and sampling methods, designing probabilistic techniques that only need to observe small subsets of packets and integrating feedback from the sidecars with functionality at the datacenter edge.

We believe that sidecars provide both an expedient and a cost effective means to program datacenter networks. Recent trends lead us to believe that even as link and router speeds evolve, advances in software processing may keep up, to provide reasonable choices for sidecars. SideCar's modest requirements open up the network to a wide set of randomized enforcement algorithms and applied control mechanisms that have only seen limited deployment to date. The applications shown here are a first step on this path.

2. CONTEXT

Switches: While switches can process and forward packets at high throughput, they provide limited programmability. They consist of a switching fabric and line cards. Packet processing is distributed across the line cards, each of which is provisioned with sufficient filtering and table lookup hardware to process the traffic entering the line card's local ports. This processing capacity, while scalable with port count, is specialized and cannot perform general processing. The control plane executes on general purpose processors; while these processors can perform arbitrary packet processing, their processing capacity is limited. These processors are slow, difficult to upgrade, and are connected to the dataplane with low bandwidth links. The number of control processors is fixed regardless of the number of line cards. Thus, switches are engineered to minimize the number of packets that are processed on the slow software path. Even if control processors could process packets at high throughput, commodity switch platforms have traditionally been closed systems that restrict customization.

Servers: Routing and switching functionality when implemented in software on servers is easily programmable[23] but can generally only process low volumes of traffic. In servers, the CPU does most of the packet processing work. Unlike line cards, commodity NICs contribute little capability beyond moving packets to and from the main memory. While servers have higher speed processors than switches, the lack of parallel, custom processing limits their scalability. Contention on the I/O and front-side buses can also bottleneck packets as they are processed by the CPU. Recent software advances such as RouteBricks [13] and PacketShader [19] have exploited architectural changes in servers to substantially improve packet processing capability. Servers have an increasing amount of parallel processing, in the form of GPUs and multi-core CPUs. Most buses have been replaced by point-to-point interconnects which scale with the number of I/O devices and CPUs. NICs now optimize for multi-processing, providing features such as multiple queues and load balancing to eliminate processor contention during I/O.

Datacenter Edge: End host software and configuration is increasingly under centralized, automatic control. Large datacenters are managed by software that images, provisions and monitors servers [21]. Network policy enforcement is distributed to end hosts, via configuring the virtual switches in hypervisors and personal firewalls in operating systems. Virtualization provides a shim layer at which new functionality can be deployed transparently; it is widely deployed in datacenters [1] and may soon become widely available in enterprise networks given the availability of hardware virtualization on most new PCs and the proliferation of new applications that exploit this [10, 12]. These advancements reduce the cost and complexity of pushing network functionality and policy enforcement to end host network stacks, suggesting new trade-offs to consider when partitioning functionality between the network, edge, and end hosts.

Related Proposals: Proposals such as OpenFlow modify the control plane of switches to expose a standard API for configuring the dataplanes, enabling experimentation [26, 27]. These APIs provide low level control over packet classification and forwarding; which in turn enables applications to perform custom control plane and dataplane processing by redirecting packets to an external server.

SideCar does not require switches to support open APIs but can leverage such APIs for steering and configuring switch forwarding tables. Here, we avoid revisiting the many applications of OpenFlow [3, 17]. These are characterized by occasional control overheads (e.g., once per flow) that invoke a centralized OpenFlow controller to make corresponding changes to forwarding tables. Instead, we explicitly focus on an orthogonal class of applications that do not benefit from OpenFlow (see §4). Such applications require programmability on the data path (e.g., all multicast packets, 5% of all packets, all block access requests on SANs) and if implemented with OpenFlow would result in sending large volumes of traffic to the OpenFlow controller for processing.

3. SideCar DESIGN

SideCar provides network designers with programmable in-network packet monitoring and rewriting (see Figure 1).

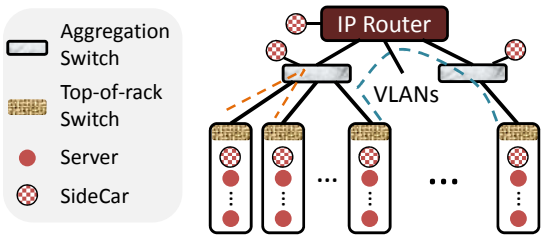

Figure 1: Extending a datacenter network with sidecars.

These programmable resources reside in sidecar processors, which lie off the standard packet forwarding path. To direct packets to sidecars, SideCar steers (or copies) subsets of packets traversing the switch.

SideCar employs two kinds of steering; the underlying redirection primitives are simple and widely supported at line rates (e.g., VLAN tagging [28], IP-IP encapsulation [16]). In *marked* steering, each switch redirects all packets with a special mark to the sidecars. Packets can be *direct* steered by sending packets to a specific sidecar. Marking and direct steering can be used both at the datacenter edge, i.e., shim layer in the host network stack or hypervisors, and between different sidecars to implement new types of forwarding schemes.

SideCar augments steering with sampling to enhance scalability and security. Sampling reduces the packet processing load on sidecars. Rather than fully trusting the end host software stack to perform all critical packet processing, which introduces potential vulnerabilities, or doing all such processing in sidecars, which are trustworthy but may not have enough computational power to do so, applications can use the available sidecar capacity to spot check the work of the end host, providing a probabilistic bound on misbehavior.

The choice of what to use as the sidecar ranges from commodity servers to RouteBricks or PacketShader class servers. This choice depends on the the volume of traffic transiting the switch that the sidecar is connected to and the demands of the application, i.e., what fraction of packets to observe and what to compute per packet.

3.1 Sampling and Steering

To improve assurance, packet sampling must be trustworthy, that is independent of any untrusted layers; sampling that is implemented by relying on untrusted layers to mark a subset of packets improves only scalability. Sampling primitives that are built into switches meet these requirements: such primitives, like NetFlow, sFlow, sampled port mirroring or port spanning, are supported in all switches. NetFlow [9] collects aggregate flow-level statistics for packets that it samples. Given a sample rate, sFlow [30] picks packets uniformly at random and forwards their headers to a preconfigured destination. sFlow is available on even entry-level ToR switches [20].

Port mirroring can sample full packets on switch ports, whether on a port- or VLAN-based match, and forward them to another port on that switch. Switches that implement packet mirroring in fabric hardware are limited only by the bandwidth of the outgoing port and can deliver packets with low latency. A high performance sidecar may take in all packets from the switch and itself implement more precise statistical sampling techniques such as trajectory sampling.

Either a switch or an end host can determine which packets to steer to a sidecar. There are a few ways to achieve steering. Analogous to mechanisms such as MPLS or DOA [31, 36], packets can be directly steered by tacking on an outermost routable header with the address of a target sidecar. Steering can use MAC-in-MAC encapsulation or IP-in-IP encapsulation to specify the target sidecar's address. To implement marked steering, switches are configured to redirect to a sidecar traffic that has a specific VLAN tag (works in L2) or type-of-service tag (works in L2 or L3) or a specific MAC or IP destination address. The edge or downstream sidecars can mark packets for processing at the next sidecar on the path.

3.2 Choosing an appropriate sidecar

Table 1 compares a few options for sidecars on their cost, as of July 2010, and appropriateness for network locations and application demands.

An 8-core compute blade with one GbE link costs $2000. Swapping in an optical 10Gbps NIC increases the price by about $800. A ToR switch may have 44 1 GbE down (40 attached to blades in the rack) and 4 10 GbE up ports. ToR switches are typically low-end switches (limited ACL, VLAN, multicast capability, no L3 support) and cost $3000, for a per-machine cost of $75. Note that, due to differences in volume and market segment, switches and sidecars can have different levels of markup; thus, comparing list prices only approximates a comparison of true costs. RouteBricks and PacketShader-class hardware have higher end processors and interconnects costing $4000-7000; the cost of optics for each of four 10GbE NICs brings up the cost to about $6000-9000. The performance of commodity servers is set to improve since low end processors will soon ship with better interconnects and on-die GPUs.

We see that a single commodity server is suitable as a sidecar at the ToR switch. It can process sizable fractions of the traffic transiting the ToR switch and perform moderate computations per packet. The per-port increase in cost due to the sidecar is about $50 ($70) for the ability to process 1.25% (4%*) of the packets through the ToR switch. This cost is comparable to that of a high-end ToR switch; SideCar's programmability enables it to match or exceed some of the features of such switches.

Note that a number of optimizations are possible. Applications that only require lightweight packet processing or only monitor a low volume of traffic allow sidecars to be shared between different switches, while more intensive workloads can use higher end sidecars. Performance estimates from RouteBricks and PacketShader suggest that one server can support

*A commodity server can't process more than 3-5Gbps [13]

Type of sidecar	Cost	Appropriateness		
		Location	Cost/port †	Application
8-core Commodity Server, 1 or 10Gbps NIC	$2800	@ ToR switch	$70	can process 1.25-4% of ToR's traffic
		@ Agg switch	N/A	unsuitable, can only process $<$ .1% of traffic
1 RouteBrick/PacketShader server, up to 4 10Gbps NICs	$9000	@ ToR switch	$14	can process 50% of ToR traffic
		@ Agg switch	$8	can process $\sim$ 2.6% of Agg's traffic
n-port RouteBrick/PacketShader router	$9000*$n$	@ Agg switch	N/A	can process all Agg traffic, L3 replacement

† Cost per 1 GbE ToR port, to process at least 2.5% traffic at indicated location, assuming 2:1 over-subscription.

Table 1: Comparing choices for sidecar on their cost and appropriateness given location in the network and application demands.

up to 35-40Gb/s today. A PacketShader blade when connected to 16 ToRs can process 3.1% of the packets at each of these ToR's and costs $14 per port, while one split across 4 different ToRs can process 12.5% of the packets for $56 per port. Both configurations are more cost-effective than using commodity blades.

An important special case is that of a non over-subscribed datacenter network. When one of the many recently proposed datacenter topologies (VL2, fat-tree, bcube, dcell) eliminate over-subscription in the core of the network, many applications such as monitoring, multicast support and controlling network bandwidth allocation only require programmatic control outside the network core, i.e., at the ToRs.

If necessary, however, RouteBricks or PacketShader class server can play the role of a sidecar at agg/core switches. Assuming a 1:2 over-subscription ratio, the cost increase is $8 per server to be able to process 2.6% of the traffic entering the agg/core switch; this includes the $16,000 cost of consuming four 10 GbE ports on a 160-port agg/core switch. The cost falls linearly as over-subscription factor increases. In non-oversubscribed topologies that use more switches, the per-port cost increases linearly with the number of additional agg/core switches needed to support the topology.

4. APPLICATIONS OF SideCar

By providing programmatic processing on a sampled subset of packets that are steered to a sidecar, we show how SideCar facilitates novel solutions to four pressing problems in cloud datacenters with tens of thousands of tenants and in enterprise networks.

4.1 Reachability Isolation

A major impediment in moving applications to the cloud is the ability to provide security, robustness and performance guarantees from the datacenter network that are equivalent to those in a client's dedicated infrastructure.

A particular threat vector is that any tenant on the cloud network may launch DDoS attacks or try to compromise the VMs of other tenants that are sharing the network. As a result, reachability isolation, i.e., a tenant's VM exchanges packets only with other VMs belonging to this tenant or shared services provided by the infrastructure, has become a desirable property in cloud datacenter.

In a sense, we would like to to place each tenant on a private network. Enforcing isolation by placing ACLs (e.g. VLAN or IP-based) in switch TCAMs runs out of TCAM space, as the number of ACLs scales at least linearly with the number of VMs whose traffic can transit the switch. Scaling up the policy sizes requires upgrading to more expensive switches; such switches are prohibitively expensive at the ToR level, limiting coverage to inter-ToR traffic. Likewise, supporting new types of policies, such as tenant- or user-based ones, can require replacing all switches [8]. The current solution is to place configurable packet filters in the edge hypervisors. While this prevents unwanted traffic from reaching the VMs, an attacker can force the hypervisor to waste CPU cycles processing these packets or burden the network with unwanted traffic.

SideCar leverages sampling to achieve, for significantly lower cost, the flexibility of hypervisor filtering while providing the same level of DoS protection as switch-based filters. A probabilistic guarantee of detecting policy violations, combined with an aggressive mechanism to contain attackers, suffices to limit the damage due to an attacker.

SideCar's Solution: SideCar uses sample-based auditing to enforce reachability isolation. Rather than filter packets, as firewalls at hosts and switches do, SideCar allows all traffic to enter the network and checks a uniform sampling against access policy. By itself, sampling provides only reachability isolation detection. To provide reachability isolation, we combine this with a strong response to policy violations. When violations are detected, the sidecar configures the ToR to revoke the sender VM's access to the network.

Employing such a response mechanism has the potential that an attacker may cause SideCar to revoke access of innocent nodes. Note however, that the detectors, on account of being physically separate sidecar processors, are harder to compromise. Further, we use address spoof prevention mechanisms at the edge (i.e., in ToR switches and hypervisor v-switches) to prevent an attacker from spoofing unwanted packets as if they were coming from someone else.

Sampling provides a probabilistic guarantee – with high probability, SideCar detects violations within a small number of packets. Suppose p is the probability of sampling a packet. Then the probability of detection after sending k policy-violating packets is $1-(1-p)^k$. When multiple sidecars lie on a path from the source to the destination, there are further detection opportunities. Assuming independent sampling at m SideCar switches along the path, the detection probability increases to $1-(1-p)^{m \cdot k}$. Given a sam-

pling probability of 1% and two SideCar switches, the detection probability is 99% after 227 packets. †

This packet limit applies to all destinations from a given VM; each VM can incur only a limited number of violations. This bounds the total packet leakage between colluding hosts on isolated networks and bounds the amount of host CPU and network capacity consumed on filtering unwanted packets that violate the policy.

When VMs move or new VMs are created or removed, the set of applicable policies changes. The datacenter management software handling such provisioning communicates the change to sidecars. For a short time after a policy change, stale packets in the network may be mistaken for violations. To guard against such false positives due to race conditions, SideCar accepts packets that conform to the old policy up to a timeout period after the change.

SideCar allows for highly expressive policies and scales to the size and churn in cloud datacenters. Similar to Berkeley Packet Filters [25], SideCar's policies in software can be specified over hosts, ports and applications. Checking for policy violations is embarrassingly parallel. PacketShader reports a rate of $>$ 10mpps with two GPUs on similar rules. Further, much work to speed up filtering large rule sets in the context of Berkeley packet filters [2] and IDS boxes can be leveraged in software.

4.2 Granular network resource allocation

Hypervisors allow controlled sharing of CPU and memory, but sharing the network relies on TCP's congestion control today. Any tenant can gain unbounded bandwidth share in the network through several means: using more TCP flows, using more aggressive variants of TCP, bypassing the congestion control in the guest VM's network stack or using protocols that do not respond to congestion, such as UDP. A tenant can use the extra bandwidth selfishly or use it maliciously to interfere with others on the shared links, switches or servers. In a sense, we want to provide a modicum of control on how the network bandwidth is shared among multiple tenants agnostic to what traffic the tenants may send.

Previous in-network approaches require traffic engineering features that are less common in datacenter switches and do not scale to the number of tenants in the datacenter. End-to-end congestion control is free from scaling limits due to switch hardware and avoids the fragmentation of network bandwidth that static reservations schemes suffer from. However, inferring congestion signals end to end (like TCP does) reacts slowly and causes suboptimal performance when all traffic is bundled through a small number of congestion controlled tunnels.

SideCar's Solution: SideCar improves edge-based congestion control by using sidecars to provide XCP [22]-like explicit feedback. Suppose all traffic from the tenant goes via congestion controlled hypervisor to hypervisor tunnels. Each VM is configured with a weight, based on size of the VM or other policy. Using feedback from sidecars on the path, rate limiters at the send hypervisors learn their max-min share of the network bandwidth. ‡

By providing explicit feedback, sidecars enable the send hypervisors to quickly yet stably converge to their appropriate shares. Specifically, the sidecars along the path sample uniformly from all packets to estimate the amount of spare bandwidth available on the bottleneck link and the identities of the hypervisors that are using the link. By passing these values back to the send hypervisors, each hypervisor can make a judicious choice of adapting its share. The increase and decrease rules while analogous to XCP, have to be modified since only a sample of all packets are observed and since feedback is not issued per-packet. We defer the details to future work.

4.3 Scalable, programmable multicast

Multicast can improve the performance of many abstractions for building large scale systems, such as consensus [4], data replication [5, 15] and mass VM start up [24]. However, the use of native IP multicast has been limited in enterprise datacenters due to concerns about its congestion stability and security [35]. Likewise, cloud providers typically do not expose multicast to their tenants.

SideCar's Solution: The datacenter edge steers packets needing multicast support to a SideCar switch on the path. SideCar maintains multicast state in the sidecars. For each group whose traffic transits through a SideCar switch, the sidecar of that switch maintains a list of the switch ports having participants in the group. Upon receiving multicast packets, a sidecar replicates the packet as necessary and forwards it out the other ports.

Multicast primitives can be built using SideCar that avoid the security and congestion implications of native IP multicast [11]. Such out-of-band support for multicast is similar to Application Layer Multicast [6] but is better performing due to fewer needless copies of packets and shorter paths. Congestion control or back pressure can be done in software to improve stability.

Supporting multicast is feasible with SideCar. Even commodity servers can handle table lookup and packet replication for up to modest volumes of multicast traffic. When the fan-out of a group is large, SideCar leverages support for local multicast groups in switch hardware. By constructing a local multicast group entry consisting of the outgoing ports that this traffic should leave on, the sidecar needs to transmit just one copy of the packet and defer replication to the data plane in the switch.

4.4 Preserving hypervisor policy control despite direct I/O

To improve network performance in virtualized environments, direct I/O from guest VMs to the NIC [14] has been

† increasing from 99% after 458 packets with just one switch.

‡ Each VM gets a share of bandwidth proportional to its weight. Unused shares are proportionally allocated to VMs that need it.

recently standardized. Direct I/O avoids the overhead of passing packets through the hypervisor. However, it comes at the cost of losing the policy control (e.g., filters, rate limiters) that is currently done in the hypervisor v-switch.

SideCar provides a way to restore the policy control of the v-switch without waiting for switch support for direct I/O to become standardized, implemented and available. Consider the example of rate limiting a VMs traffic to the network. SideCar achieves this by asking the guest VM to limit its traffic. However, there is no guarantee that the guest VM, which can be arbitrarily modified by the tenant, will do so. Hence, a SideCar switch at the ToR upstream of the guest samples all traffic leaving the ToR. By sampling uniformly at random, SideCar can project from the proportion of the guest's traffic seen in the sample to check that the VM is within limit.

Consider another example of counting a VM's seek load and bandwidth utilization on a SAN, which impact the throughput of other VMs accessing disks on the same SAN [18]. SideCar expects the guest to mark and steer disk command packets to a sidecar that can track these metrics. However, a guest can deflate its request rate by only steering some of its command packets. To prevent the guest from doing so, SideCar can sample all packets and verify that all the request packets were indeed marked to be counted.

5. FINAL REMARKS

Inability to modify switches has long been the bane of network innovation. We present SideCar, an architecture that uses recent improvements in processing packets with commodity servers to provide generic packet processing on the dataplane. SideCar changes the configuration of switches but requires no software or hardware modifications at switches. By using host hypervisors, SideCar provides more expressive packet classification and marking. Sampling lets SideCar scale with low cost. We show that SideCar brings ideas from randomized sampling and explicit control to bear on pressing problems in cloud datacenter networks. As frameworks to manage cloud datacenters evolve, such joint edge/network designs become more important design patterns to achieve better performance and security at low cost.

6. REFERENCES

[1] Amazon Web Services. http://aws.amazon.com/.

[2] A. Begel, S. McCanne, and S. L. Graham. BPF+: Exploiting Global Data-flow Optimization in a Generalized Packet Filter Architecture. *ACM CCR*, 1999.

[3] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. McKeown, and S. Shenker. Ethane: Taking Control of the Enterprise. *CCR*, 2007.

[4] M. Castro and B. Liskov. Practical Byzantine fault tolerance. *ACM TOCS*, 1998.

[5] F. Chang, J. Dean, S. Ghemawat, and W. Hsieh. BigTable: A distributed storage system for structured data. *ACM TOCS*, 2008.

[6] Y. Chu, S. Rao, and H. Zhang. A Case for End System Multicast. *Proceedings of SIGMETRICS*, Oct. 2000.

[7] Cisco Systems. Traffic Anomaly Detection and Mitigation Solutions. http://www.cisco.com/en/US/prod/collateral/vpndevc/ps5879/ps6264/ps5887/prod_bulletin0900aecd800fd124_ps5888_Products_Bulletin.html.

[8] Cisco Systems. TrustSec. http://www.cisco.com/en/US/netsol/ns1051/index.html.

[9] B. Claise. RFC3954: Cisco Systems NetFlow Services Export Version 9, 2004.

[10] T. Das, P. Padala, V. Padmanabhan, R. Ramjee, and K. G. Shin. LiteGreen: Saving Energy in Networked Desktops Using Virtualization. *USENIX ATC*, 2010.

[11] S. Deering and D. Cheriton. Multicast routing in datagram internetworks and extended LANs. *ACM TOCS*, 1990.

[12] C. Dixon, H. Uppal, D. Brandon, A. Krishnamurthy, and T. Anderson. An End to the Middle. In *(under submission)*, 2010.

[13] M. Dobrescu, N. Egi, K. Argyraki, B. Chun, and K. RouteBricks: Exploiting parallelism to scale software routers. *SOSP*, 2009.

[14] Y. Dong, Z. Yu, and G. Rose. SR-IOV Networking in Xen: Architecture, Design and Implementation. In *WIOV*, 2008.

[15] S. Ghemawat, H. Gobioff, and S.-T. Leung. The Google file system. *SIGOPS OSR*, 2003.

[16] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta. VL2:Â A scalable and flexible data center network. *SIGCOMM*, 2009.

[17] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker. NOX:Â towards an operating system for networks. *ACM SIGCOMM CCR*, 2008.

[18] A. Gulati and C. A. Waldspurger. PARDA : Proportional Allocation of Resources for Distributed Storage Access. In *FAST*, 2009.

[19] S. Han, K. Jang, K. Park, and S. Moon. PacketShader: a GPU-Accelerated Software Router. *SIGCOMM*, 2010.

[20] Hewlett-Packard. HP ProCurve 2910al Switch Series. http://h10146.www1.hp.com/products/switches/HP_ProCurve_2910al_Switch_Series/overview.htm/.

[21] M. Isard. Autopilot. *SIGOPS OSR*, 2007.

[22] D. Katabi, M. Handley, and C. Rohrs. Congestion control for high bandwidth-delay product networks. *SIGCOMM*, 2002.

[23] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek. The Click modular router. *ACM TOCS*, 2000.

[24] H. A. Lagar-Cavilla, J. A. Whitney, A. M. Scannell, P. Patchin, S. M. Rumble, E. de Lara, M. Brudno, and M. Satyanarayanan. SnowFlock. *EuroSys*, 2009.

[25] S. McCanne and V. Jacobson. The BSD packet filter: A new architecture for user-level packet capture. *USENIX Winter*, 1993.

[26] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. OpenFlow: Enabling innovation in campus networks. *ACM CCR*, 2008.

[27] J. C. Mogul, Praveen Yalagandula, J. Tourrilhes, R. McGeer, S. Banerjee, T. Connors, and P. Sharma. API Design Challenges for Open Router Platforms on Proprietary Hardware. *HotNets*, 2008.

[28] J. Mudigonda, P. Yalagandula, M. Al-Fares, and J. Mogul. SPAIN: COTS Data-Center Ethernet for Multipathing over Arbitrary Topologies. *NSDI*, 2010.

[29] R. Pan, B. Prabhakar, and A. Laxmikantha. QCN : Quantized Congestion Notification. IEEE 802.1Qau Presentation, 2007. http://www.ieee802.org/1/files/public/docs2007/au-prabhakar-qcn-description.pdf.

[30] P. Phaal and M. Lavine. sFlow Version 5. 2004.

[31] E. Rosen, A. Viswanathan, and R. Callon. RFC3031: Multiprotocol Label Switching Architecture. 2001.

[32] SourceFire. 3D Sensor. http://www.sourcefire.com/products/3D/sensor.

[33] T. Spalink, S. Karlin, L. Peterson, and Y. Gottlieb. Building a robust software-based router using network processors. *SIGOPS OSR*, 2001.

[34] D. Tennenhouse and D. Wetherall. Towards an active network architecture. *ACM SIGCOMM*, 1996.

[35] Y. Vigfusson, H. Abu-Libdeh, M. Balakrishnan, K. Birman, and Y. Tock. Dr. Multicast: Rx for Data Center Communication Scalability. *LADIS*, 2008.

[36] M. Walfish, J. Stribling, M. Krohn, and H. Middleboxes no longer considered harmful. *OSDI*, 2004.