In-packet Bloom filter based data center networking with distributed OpenFlow controllers

Carlos A. B. Macapuna, Christian Esteve Rothenberg, Maurício F. Magalhães
School of Electrical and Computer Engineering
University of Campinas, Brazil
Email: {macapuna, chesteve, mauricio}@dca.fee.unicamp.br

***Abstract*—This paper discusses a novel data center architecture based on load-balanced forwarding with in-packet Bloom filters enabled by two support services that distribute the directory and topology state of OpenFlow controller applications. By deploying an army of Rack Managers acting as OpenFlow controllers, the proposed architecture promises scalability, performance and fault-tolerance. We conjecture that packet forwarding itself may become a cloud internal service implemented by leveraging cloud application best practices such as low-latency, highly-available, key-value storage systems. Moreover, we contribute to demystifying the argument that the centralized controller model of OpenFlow networks is prone to a single point of failure and show that direct network controllers can be physically distributed, yielding a "sweet spot" between fully distributed and centralized networking paradigms.**

I. Introduction

Driven by unprecedented scale, cost, and control objectives, data center networks (DCN) are undergoing an intense transformation [1], [2]. Cloud infrastructure providers are striking for optimized solutions, a multidisciplinary daunting task that has spurred creative designs like shipping-container-tailored designs [3] or re-thinking the flatness of Ethernet MAC addresses to embody network locations [4]. From a networking perspective, DCNs have very special requirements compared to traditional enterprise or inter-domain networks, e.g., agile VM provisioning, unpredictable traffic patterns, etc. [2]

With the advent of software-defined networks as proposed by OpenFlow [5], DC networking can be turned into a software problem, tractable by the same developers of the so-called warehouse-scale computers [1]. Basically, OpenFlow specifies a standard way for controlling packet forwarding decisions in (remote) software while keeping the hardware vendors in charge of the device implementation. This separation of concerns leads to a promising combination of the programmability of PCs with line-speed commercial networking hardware.

Motivated by this context and inspired by recent developments [2], [4], [6], we propose a forwarding service based on encoding Valiant load-balanced source routes (cf. VL2 [2]) into in-packet Bloom filters (cf. LIPSIN [6]) carried in Ethernet fields re-written at Top-of-Rack (ToR) switches (cf. Portland [4]). Our previous work [7] introduced the core principles of data center networking with in-packet Bloom filters (iBF), focusing on the evaluation of the iBF forwarding data structure and our claims of being able to circumvent false positives. In this paper, we present the SiBF (Switching with in-packet Bloom filters) data center architecture along a new version of our testbed environment, which moves away from a centralized controller implementation and introduces an army of Rack Managers (RM), one per rack and acting as OpenFlow controllers, promising scalability, performance, and fault-tolerance by maintaining the globally required state (topology and VM directory) in a key-value data store — in spirit of (or even re-using) low-latency distributed storage services commonly available in cloud data centers as application support functions (e.g., Amazon Dynamo).

We believe SiBF makes the following contributions:

1) it proposes a scalable forwarding service that is amenable to existing (OpenFlow-capable) networking hardware, does not require endpoint modifications, is self-configurable, and can be offered as an alternative forwarding service in parallel to other Ethernet flavours.
2) it shows how an OpenFlow controller application can be easily implemented in a distributed fashion, yielding a "sweet spot" between fully distributed and centralized networking paradigms.
3) it argues that packet forwarding itself may become a data center internal service tailored to the application needs and leveraging cloud programming practices.

The balance of the paper is as follows. Section 2 introduces the principles and building blocks of SiBF. Section 3 describes the prototype implementation and the testbed environment. We discuss the key features in Section 4, compare to related work in Section 5, and conclude with an outlook in Section 6.

II. Overview of SiBF

Basically, SiBF uses IP addresses solely for VM identification and provides a scalable, load-balanced packet forwarding service based on encoding (randomly chosen) source routes into a 96-bit iBF carried in the L2 fields. iBF flow mappings are dynamically installed at first hop switches (i.e., ToRs) by each rack's OpenFlow controller — the Rack Manager.

• **Topology:** We assume a 3-tier ($CORE$, $AGGR$ and ToR) multi-rooted tree topology due to its appealing properties for DCN, as such, it offers multiple paths between any pair of servers, facilitates load balancing and is easily implemented through commoditized switches. However, iBF-based forwarding works on arbitrary graphs and other scale-out topologies could be considered (e.g., DHT rings, Hypercubes), as long as they offer large path diversity and low diameter.

• **Separating names from locations:** With IP addresses used as VM identifiers (potentially along a VLAN tag to enable multi-tenancy and overlapping IP address spaces), pure layer 2 connectivity is granted by a revisited iBF-based source-routing capable Ethernet layer at intermediate switches. This approach aims to minimize the forwarding table of switches and disaggregate the semantics present in the IP protocol.

• **Source explicit routing:** Taking advantage of the small diameter of the DCN topologies, source routing iBFs are generated at RMs and contain the Bloomed MAC IDs of $CORE$, $AGGR$ and ToR switches. Intermediate switches remain stateless and only need to "query" for iBF presence of the neighboring switches. Knowing the topology, *false-positive-freeness* can be achieved by RMs having iBFs tested prior to their use; maintaining a ToR reachability matrix filled only with false-positive-free iBFs (one per available path).[1]

• **Logically centralized direct network control with distributed application state:** Rack Managers (RM) implement the routing logic by installing flow states at ToRs to rewrite the Ethernet fields with iBFs (at the source) and legacy MACs (at the destination). In addition, RMs maintain a distributed NoSQL (Not only SQL) database holding (1) the topology information (link tuples), and (2) the VM directory ($VM_{id} \rightarrow ToR_{dst}$). RMs independently write their discovered events (e.g., VM x active behind ToR y, switch join/leave, link up/down) and continuously read (and cache) the network link map and the VM locations.

• **Load balancing via path randomization:** Path multiplicity is exploited by virtue of *oblivious routing* (i.e., traffic independent randomized routing) as RMs randomly pick iBF to reach the destination. By not relying on flow-hashing, iBF-based Valiant Load Balancing (VLB) avoids ECMP inefficiencies like hash collisions or the 16-way limitation.

• **Unmodified end-points and plug & play:** Legacy servers and applications are supported off-the-shelf and auto-configuration of hosts and switches. The latter provided by a Role Discovery Protocol based on extending the LLDP. [7]

• **Design to cope with failures:** At cloud scale, component failures are common. SiBF is resilient to server and network failures as it does not rely on any single node. Furthermore, it provides data flow protection similar to MPLS fast re-routing.

A. False-positive-free forwarding

The major differential factors of iBF-based forwarding is not requiring any per end-host (or per VM) state in the intermediate switching layer ($AGGR$ and $CORE$) and avoiding encapsulation by re-using the Ethernet MAC fields. However, the well-known caveat of Bloom filters is the possibility of false positives (i.e., claiming an element to be present when it was not inserted). In its original, data-oriented multicast application [6], iBF-based forwarding meant opening a new vector in the design space of multicast routing, namely potential efficiency penalties due to unnecessary packet duplications.

[1]NS-3 simulations on large scale topologies show that the price in reduced path multiplicity is less than 1%, as expected by the false positive rate of a $n = 96$ BF with only $n = 3$ elements. See [7] for details and alternatives.

The problem of such packet duplications in the dense connected data center switching fabric is the risk of consecutive falsely forwarded packets not being discarded (due to lacking of a matching entry at next hops) and ending up in a loop. Though the chances are very low, we need to guarantee forwarding completeness, i.e., loop-free delivery of packets to the intended destination. Our strategy is to exploit the notion of the power of choices in multiple iBF representations, i.e., one iBF for each available network path. With knowledge of the topology, RMs can maintain a ToR destination matrix filled only with false-positive-free iBF paths. For instance, in a 3-tier 5-stage Clos topology with 48-port CORE and AGGR switches, false-positive-free means that, on average, instead of enabling all 96 multiple paths for load balancing, "only" 94 can be made available (cf. [7]). A second strategy to circumvent false positives is to use d different sets of hash functions (similar to the d-candidate LITs in [6]).

B. Topology and directory services with an army of RM

Our architecture proposes two services (topology and directory) to provide scalability and fault-tolerance in the logically centralized direct network control approach of an OpenFlow network. Both services provide the network-wide information necessary for the routing and control functions performed by the RM behind each rack. This army of RMs is required to keep manageable both the amount of flow initiation requests and the total switch-to-controller traffic in a data center network of potentially tens of thousands of devices. In addition, current OpenFlow-based neighbour discovery relies on controller-generated LLDP messages that constitute a considerable burden for a reduced number of controllers. Sharing the control of AGGR and CORE switches among a subset of independently working RMs addresses this issue.

A non-trivial point is the correct inference of the topology of a multi-rooted switching tree and the role (layer) each switch plays in the network (i.e., $CORE$, $AGGR$ or ToR). This task is performed by our Role Discovery Protocol (TreeDiscovery RM application) and handled to the Topology Service (TS), recalling that timely knowledge of topology is a prerequisite for any source route approach (e.g., Portland [4]). The TS is responsible for storing the network topology and its consistent distribution to the RMs. $CORE$ and ToR switches get "Bloom" flow entries installed consisting of only k bits set in the Ethernet src and dst fields as determined by k independent hashes of the neighbouring switch MAC. [2]

Finally, RMs need to know the location of destination servers, i.e., the destination ToR to reach the VM with a given IP address. This mapping is performed by the distributed Directory Service (DS) and updated by RMs according to the discovery of VM based on host-initiated ARP requests. An ARP replier module in the RM eliminates the network flooding caused by the propagation of ARP packets.

The overall state of the information collected by the TS and DS is stored in a highly-available NoSQL database of

[2]In practice, it suffices to generate the k values from a pseudo random combination of the lower 24-bit MAC address

the type key-value. This approach extends the concept of direct network control of OpenFlow and addresses the single point of failure problem of centralized control. We keep a *logically* centralized control, where the state of the topology and directory is maintained globally, and physically distributed controllers directly act on a subset of switches.

III. Testbed implementation

The Rack Manager (RM) and its modules Directory Service (DS), Topology Service (DS), ARP replier, and Topology Discovery, are implemented as applications on top of the NOX controller [8] (see Fig. 1). NOX's programmatic interface is built upon *events*, triggered by NOX core components, thrown by user-defined applications, and generated directly from OpenFlow protocol messages. Enabling iBF-based forwarding in OpenFlow switches requires flow matching on arbitrary wildcarded L2 bitmasks (i.e., match if k bits are set to 1 in Eth SA/DA). Official support is expected in upcoming versions and required trivial changes in only two lines of code[3] of the reference implementation.

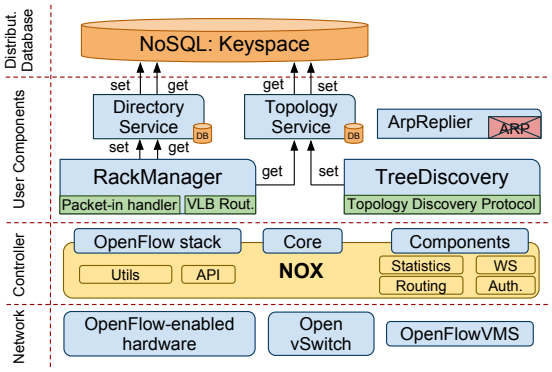


Fig. 1. SiBF system architecture

The testbed consists of a series of physical nodes hosting the VM instances of RMs, key/value store nodes (DB), OpenFlow switches, and end servers. Fig. 2(a) shows the prototype implementation within a physical server and depicts how the testbed scales by interconnecting additional servers. Topologies are configured with OpenFlowVMS [5], which default scripts we have extended to distribute networked VMs (using QEMU and VDE+SSH) across different physical machines, enabling a quick deployment of target topologies, automated bootstrapping of virtual nodes and switches, IP configuration, data-path creation, modules start-up, and the connectivity to the controllers. We use KeySpace [9] as the NoSQL database, and integrate it with the RM via a Python API.

In addition, we have a second testbed environment based on Mininet [10]. Mininet is a simple and scalable platform for virtual networking, which works in the operating system kernel, creating processes and not complete virtual machines to host the OpenFlow switches, offering a far superior performance compared to full virtualization itself. This way, Mininet allows to easily perform tests for various network topologies, with a more significant number of switches and end-servers per physical server. Fig. 2(b) shows a our Mininet testbed environment hosted in a single machine a total of 8 $CORE$, 16 $AGGR$, 32 ToR, and 192 end nodes. The practical limitation on the total number of switches (56) is due to the current alpha version of Mininet not supporting multiple controllers, which makes impossible to keep scaling out following a distributed control of OpenFlow as in our testbed of Fig. 2(a). All in all, both testbeds use the same code and allow to quickly boot and assess different topologies by simply modifying a few parameters such as number of switches, servers and connected interfaces.

[3]function `flow_fields_match` in `udatapath/switchflow.c`

IV. Discussion

In this section, we discuss the key features of the data center forwarding service provided by SiBF.

A. State requirements

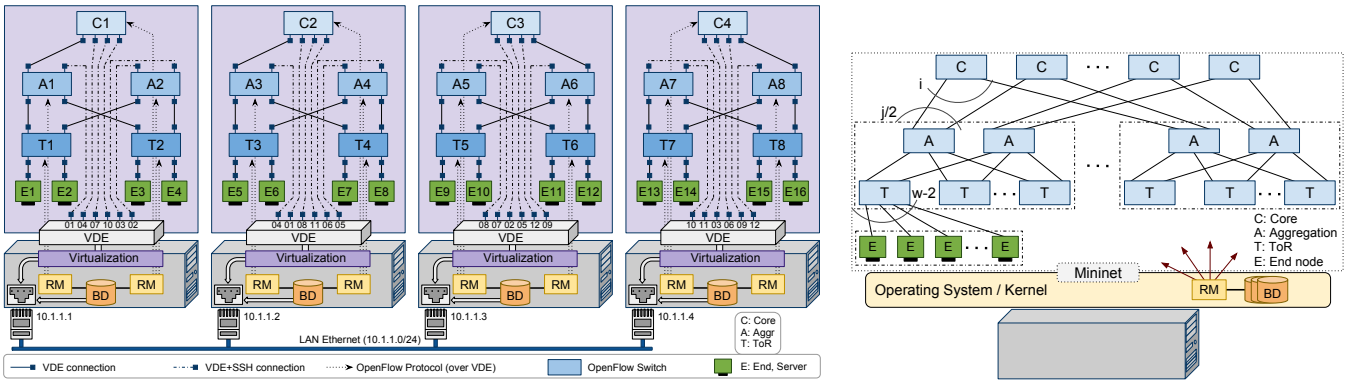
Consider a three-tier network topology, with ToRs connected to 20 servers and 2 AGGRs with links of 1 and 10 Gbps, respectively. The AGGR switch ports (p_1) are used to connect to $p_1/2$ ToRs and $p_1/2$ COREs with p_2 links. Depending on the exact values of p_1 and p_2, the interconnection matrix can be scaled from e.g., 3,000 ($p_i = 24$) to 100,000 ($p_i = 144$) physical nodes. Due to the strict source routing approach, flow table requirements in SiBF are minimal and constant in the COREs and AGGRs switches, i.e., only one entry per interface as shown in Table I. Further scaling the network does not affect the number of entries flow in the switches that is constant and equal to the number of neighboring switches. At ToRs, the amount of flow entries grows with the number of simultaneous outbound flows (assumed 5 per VM) plus a constant amount of entries (1 per VM) in order to perform the MAC re-writing and correct delivery of inbound packets. Moreover, we can move the flow state from the ToRs to the end servers by relying on OpenFlow-enabled virtual switches (cf. [11]).

TABLE I
State requirements in terms of entries at switches. Assuming 10 flows/VM and 20 VM/server: 5 inbound and 5 outbound.

Physical hosts	23.040			103.608		
Racks	1152			5184		
Aggr. Switches	96 ($p_1 = 48$)			144 ($p_1 = 144$)		
Core Switches	24 ($p_2 = 96$)			72 ($p_2 = 144$)		
	VL2	Portland	SiBF	VL2	Portland	SiBF
Entries at ToR	1672	2400	2400	5800	2400	2400
Entries at AGGR	1272	48	48	5400	144	144
Entries at CORE	1272	96	96	5400	144	144

B. Timeliness

Each RMs is a standalone application running in one of the rack servers directly connected to the OpenFlow ToR switch they control. Hence, flow requests are handled locally (at rack-level), with expected flow setup times in the order of 10s of ms. Candidate routing iBF for each destination ToR are pre-computed and can be used without further delays, while VM directory mappings are continuously re-freshed.



(a) OpenFlowVMS and Qemu/VDE

(b) Mininet

Fig. 2. SiBF testbed environment.

C. Fault tolerance

OpenFlow controller: In the failure event of the RM controller, new flow requests cannot be handled unless a slave RM takes over. Distributing the centralized control of OpenFlow among an army of autonomous RMs reduces the scope of a controller failure to a rack only. Once the necessary mechanism to allow back-up OpenFlow controllers to take over is fully worked out, flow requests can be handled by any other stand-by RM (e.g., from a neighbouring rack). Ongoing data traffic is not affected by the failure of the controller, which can be put back to normal operation after rebooting and recovering all relevant state from the TS and DS.

OpenFlow switches: When a switch fails, the current traffic through that switch is interrupted until the iBF mappings at source ToRs are changed (see below). New flows are routed through alternative paths once the discovery protocol updates the network topology by removing the link tuples of the unavailable switch. With iBF-based forwarding we can implement a very effective protection strategy similar to MPLS-based IP fast re-route. The approach consists of installing back-up flow entries at ToRs for each new active flow. This back-up flow entry is set with a lower priority and maps to a link disjoint iBF path. In the event of a switch failure on the path of the primary iBF flow entry (or upon a congestion event), the RM simply increases the priority value of the back-up flow entry and outbound packets hitting the source ToR will be re-written with the new iBF and carried over a new (parallel) path to the destination.

NoSQL database: If the database service fails, the RM can still work based on the replicated information stored in cache, but changes in the network will not be updated globally. The Keyspace database [9] implementation (similar to any cloud NoSQL appliance) is resistant to failures of any database nodes as long as a minimum of two are keep alive to ensure consistency of the underlying Paxos algorithm.

D. Load Balance

We have implemented the Valiant Load Balacing (VLB) scheme (cf. VL2 [2]) by randomly picking the iBF for each new flow request. Note that this load balancing strategy is more powerful — as it has more waypoints — than per-flow static hashing approaches like ECMP. Recent work [12] discusses how this can be specially critical in long lived data-intense flows within the data center. Additionally, switches today only support up to 16-way ECMP. [4]

We have compared the link usage of SiBF with the implementation of a vanilla Spanning Tree Protocol (STP) over the same topology. We have conducted 100 runs of the experiment consisting of generating traffic from all-to-all parties in our testbed. As expected, with STP some network links are heavily used and others underutilized, reaching up to 350% in the maximum normalized link utilization compared to an ideal traffic distribution. In contrast, VLB in SiBF showed a variation of only 20% (max and min values) around the average link utilization. An open question is how to deal with problematic (underperforming) elephants flows. While we have not worked out any scheduling system that adaptively re-routes this traffic to keep the observed performance high (cf. Hedera [12]), any central scheduler algorithm with global knowledge of active flows can be implemented by accordingly changing the iBF mapping at originating edge ToRs.

E. Enabling differentiated forwarding services

Given the zero state requirements at intermediate switches, we can easily offer iBF forwarding in the data center without consuming precious switch resources. We envision that packet forwarding in data centers may have multiple flavours depending on the specific application requirements and potentially on the associated SLA. For instance, map-reduce type of applications may be offered a highly randomized path-diversified forwarding layer based on iBFs while traffic from other services/VMs may be delivered using traditional Ethernet forwarding. Software-defined networking as proposed by OpenFlow enables this type of multi-tenant environments, where heterogeneous applications can operate under differentiated control planes over a shared physical infrastructure.

[4] According to [2], 256-way ECMP is on the production line.

V. Related Work

Traditional Ethernet switching is unsuitable for the large-scale and high-performance computing needs of cloud data centers. Industry efforts have been undertaken towards Data Center Ethernet extensions to provide QoS, enhanced bridging (IEEE 802.1 DCB), multi-pathing (IETF TRILL), Fibre Channel support, and additional Converged Enhanced Ethernet (CEE) amendments (IEEE 802.1Qaz/Qbb).

VL2 [2] offers a scalable virtual layer 2 based on Valiant load-balanced source routing with up to three IP-in-IP encapsulations. A Directory System (DS) provides node lookups, updates and network maps. The DS achieves high performance (low latency and high availability) via two layers of replicated servers, one optimized for reading and mapping (Directory Servers) and one optimized for writing (Replicated State Machine), which uses the Paxos consensus algorithm to ensure consistency. In addition to Paxos, our TS/DS implementation based on Keyspace [9] uses similar optimization methods for *dirty* reading and *safe* writing.

Forwarding in Portland [4] is based on position-based pseudo MAC (PMAC) addresses assigned to end nodes to provide scalability without changing the Ethernet address format. It defines a location discovery protocol (LDP) to infer the exact position of switches in the multi-rooted tree topology and store this information in a logically centralized network infrastructure management service called Fabric Manager (FM). The FM maintains a state of network topology and a mapping between the real MAC address of the node and its PMAC. Both the LDP and the FM have similar characteristics to our architecture, for example, prior knowledge of the positions of switches for routing and maintaining a global state of the network. iBF forwarding differs in that it natively supports load balancing over arbitrary topologies and depends on a fairly simpler switch role discovery mechanism. Hedera [12] handles the problem of traffic randomization and flow scheduling in the Portland. Similar flow monitoring and scheduling methods could be beneficial and applicable in SiBF.

There are other remarkable proposals that have influenced the course of our work. Rack Managers follow the rationale behind the 4D [13] architecture on refactoring the control and data planes. Ripcord [10] is a platform for experimenting data center networking approaches, motivated the development of Mininet for theses purposes and advocates for multiple forwarding services being delivered on a cloud application basis over the same infrastructure.

Finally, Hyperflow [14] is, to our best knowledge, the only work so far also tackling the issue of distributing the OpenFlow control plane for the sake of scalability. In contrast to our approach based on sharing the application state in a globally available data store — well suited for our routing purposes, HyperFlow proposes to push (and passively synchronize) all state (controller relevant events) to all controllers. This way, each controller thinks to be the only controller at the cost of requiring minor modifications to applications.

VI. Conclusions and Future Work

This paper adds to the body of work towards scale-out strategies (i.e., commodity hardware, cost-effective switching topologies) extending the next frontier in DCN from "commoditization" to "customization."

SiBF forwarding service requires minimal in-network state, useful to offer unfettered, application-specific communication services. Making a "Bloom" use of the 96 bits of MAC fields, we avoid FIB table explosion and encapsulation overhead, while conserving the nice PnP properties of Ethernet and excellent path-multiplicity. Due to the army-like deployment of RMs, all necessary flow setup computations are contained within each rack, requiring only globally available and timely directory and topology information, which greatly contributes to scalability and fault-performance.

In future work, we intend to investigate anycast controller communications, iBF-based VM mobility, and contribute to the fundamental tensions of OpenFlow, e.g., what functionality is kept on the switch and what can be moved to the controllers. Regarding "traffic engineering," the iBF-based fine control over network paths opens the door to enhanced load balancing beyond VLB (e.g., flow re-routing based on congestion events and network probing/monitoring) and novel middlebox transversal mechanisms. Another avenue for future work relates to extending iBF forwarding across geo-distributed data centers.

References

[1] L. A. Barroso and U. Hölzle, *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines*, M. D. Hill, Ed. San Rafael, CA, USA: Morgan & Claypool, 2009.

[2] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta, "VL2: a scalable and flexible data center network," in *SIGCOMM '09*, Barcelona, Spain, 2009.

[3] C. Guo, G. Lu, D. Li, H. Wu, X. Zhang, Y. Shi, C. Tian, Y. Zhang, and S. Lu, "Bcube: a high performance, server-centric network architecture for modular data centers," in *SIGCOMM '09*.

[4] R. Niranjan Mysore, A. Pamboris, N. Farrington, N. Huang, P. Miri, S. Radhakrishnan, V. Subramanya, and A. Vahdat, "Portland: a scalable fault-tolerant layer 2 data center network fabric," in *SIGCOMM '09*.

[5] OpenFlow Switch Consortium, http://www.openflowswitch.org/.

[6] P. Jokela, A. Zahemszky, C. E. Rothenberg, S. Arianfar, and P. Nikander, "LIPSIN: line speed publish/subscribe inter-networking," in *SIGCOMM '09*. Barcelona, Spain: ACM, 2009.

[7] C. E. Rothenberg, C. A. B. Macapuna, F. Verdi, M. F. Magalhes, and A. Zahemszky, "Data center networking with in-packet Bloom filters," in *SBRC 2010*, Gramado, Brasil, may 2010.

[8] "NOX: An OpenFlow Controller," http://noxrepo.org/wp/.

[9] M. Trencseni and A. Gazso, "Keyspace." Scalien. [Online]. Available: http://scalien.com/whitepapers

[10] M. Casado *et al.*, "Ripcord: A modular platform for data center networking," Tech. Rep. UCB/EECS-2010-93, Jun 2010. [Online]. Available: http://www.eecs.berkeley.edu/Pubs/TechRpts/2010/EECS-2010-93.html

[11] B. Pfaff, J. Pettit, T. Koponen, K. Amidon, M. Casado, and S. Shenker, "Extending networking into the virtualization layer," in *Proc. of workshop on Hot Topics in Networks (HotNets-VIII)*, 2009.

[12] M. Al-Fares, S. Radhakrishnan, B. Raghavan, N. Huang, and A. Vahdat, "Hedera: Dynamic flow scheduling for data center networks," in *7th ACM/USENIX - NSDI*, San Jose, CA, Apr. 2010.

[13] A. Greenberg, G. Hjalmtysson, D. A. Maltz, A. Myers, J. Rexford, G. Xie, H. Yan, J. Zhan, and H. Zhang, "A clean slate 4D approach to network control and management," *SIGCOMM CCV*, 2005.

[14] A. Tootoonchian and Y. Ganjali, "HyperFlow: A distributed control plane for openflow," in *INM/WREN'10*, San Jose, CA, Apr. 2010.