Data center networking with in-packet Bloom filters

Christian Esteve Rothenberg,[1] **Carlos A. B. Macapuna,**[1]
Fábio L. Verdi,[2] **Maurício F. Magalhães,**[1] **András Zahemszky**[3]

[1]FEEC – University of Campinas (Unicamp)
Caixa Postal 6101 – 13.083-970 – Campinas – SP – Brazil

[2]Federal University of São Carlos (UFSCar)
Campus Sorocaba – SP – Brazil

[3]Ericsson Research NomadicLab / HIIT
Helsinki, Finland

{chesteve, macapuna, mauricio}@dca.fee.unicamp.br

verdi@ufscar.br, andras.zahemszky@ericsson.com

Abstract. *This paper describes a networking approach for cloud data center architectures based on a novel use of in-packet Bloom filters to encode randomized network paths. In order to meet the scalability, performance, cost and control goals of cloud infrastructures, innovation is called for at many areas of the data center environment, including the underlying switching topology and the packet forwarding paradigms. Motivated by the advent of high-radix, low-cost, commodity switches coupled with a substrate of programmability, our proposal contributes to the body of work re-thinking how to interconnect racks of commodity PCs at large. In this work, we present the design principles and the OpenFlow-based testbed implementation of a data center architecture governed by Rack Managers, which are responsible to transparently provide the networking and support functions to cost-efficiently operate the DC network. We evaluate the proposal in terms of state requirements, our claims of false-positive-free forwarding, and the load balancing capabilities.*

1. Introduction

With the advent of Internet cloud services, the underpinning data center networks (DCN) have become a matter of intense research to raise their scale, performance, and cost-efficiency to unprecedented levels [Greenberg et al. 2009a]. In order to meet these goals without sacrificing service quality, innovation is called for at many areas of the data center environment, including the hosting infrastructure itself (e.g., energy management, wiring) and the network and system engineering (routing, virtualization, monitoring, etc.)

Recent research in re-architecting data center networks has spurred creative designs to interconnect servers at large, including shipping-container-tailored designs with servers acting as routers and switches as crossbars (BCube [Guo et al. 2009]), commoditized fat-tree topologies [Al-Fares et al. 2008], forwarding on position-based pseudo MAC addresses (Portland [Niranjan Mysore et al. 2009]), or load-balanced switching clouds providing the illusion of a single virtual layer 2 (VL2 [Greenberg et al. 2009b]). Traditional DCN architectures consist of a tree of networking elements (L2/L3 switches)

with progressively more specialized and expensive equipment moving up the network hierarchy. Unfortunately, even when scaling up, resulting topologies may only offer a fraction of the aggregate capacity available at the end hosts, with reported over-subscription rates as high as 1:240 [Greenberg et al. 2009a].

While diverging in their architectural approach (e.g., server-centric vs. network-centric), every next generation DCN design proposal aims at providing a scalable, cost-efficient networking fabric to host Web, cloud and cluster applications. Many of these applications require bandwidth-intensive, one-to-one (e.g., video coding/streaming), one-to-several (e.g., distributed file systems), one-to-all (e.g., application data broadcasting), or all-to-all (e.g., MapReduce) communications among servers. Non-uniform bandwidth among data center nodes complicates application design and limits the overall system performance, turning the inter-node bisection bandwidth the main bottleneck in large-scale DCNs. Recent data center traffic characterization studies [Benson et al. 2009, S. Kandula and Patel 2009] have shed some light on the nature of DCN traffic, concluding that traffic demands are unpredictable and highly bursty, two factors that hamper traditional traffic engineering solutions (e.g., VLAN QoS). A closely related issue is the necessity of avoiding the fragmentation of resources (i.e., available servers and network paths) throughout IP subnets and VLAN domains. In highly virtualized cloud DCs, network *agility* is key to achieve high levels of server utilization and let virtual machines (VM) be dynamically instantiated (and live migrated) in any available physical server. An example of a job demanding agility might be to accommodate VMs on-demand to host Web services dedicated to the World Cup football championship during two months. In order to have an agile and unfragmented DCN, ideally, the underpinning interconnection fabric should behave like a big Ethernet domain that exploits the path diversity and scales sub-linearly to the number of addressable endpoints. Unfortunately, the flat routing nature of Ethernet does not scale beyond certain boundaries due to the lack of aggregation capabilities, the constraints of MAC-based forwarding tables, and the ARP flooding.

In this paper, we present SiBF (Switching with in-packet Bloom filters), a DCN proposal motivated by the changes in networking driven by the advent of high-radix, low-cost, commodity switches coupled with a substrate of programmability (e.g., OpenFlow [McKeown et al. 2008]). Our design borrows characteristics from a few new generation DCN designs, for instance building upon proven interconnection topologies (e.g., Clos networks) and reliance on logically centralized controllers in spirit of 4D [Greenberg et al. 2005]. Compared to related work, our key difference is the forwarding approach based on an in-packet Bloom filter (iBF) expedited by what we call a new entity in the data center: the Rack Manager (RM). The RM follows a direct network control approach to transparently provide the networking functions (address resolution, route computation) and support services (topology discovery, monitoring, optimization) to unmodified (physical and virtual) servers behind Top-of-Rack (ToR) switches.

Forwarding in SiBF takes on the idea of moving network state to the packet headers in form of a compact, multicast-friendly source route representation amenable to low-cost, high performance networking gear [Jokela et al. 2009]. Basically, SiBF efficiently interconnects any pair of communicating nodes within the DCN by compactly representing the packet's source route into a Bloom filter carried in the Ethernet MAC fields. Design goals include conserving the IP semantics and yield a false-positive-free

forwarding fabric by leveraging DC's topological properties and exploiting the multiple paths available. We address the issue of having a system with two mutually conflicting requirements: 1) flat (non-hierarchical) L2 addresses, and 2) aggregation. Our approach is to open another vector of the design space, namely potential efficiency penalties due to false positives resulting in some packets using unnecessary links. The proposed solution makes better use of the 96-bit space of source and destination MAC fields, avoiding thereby encapsulation and shim-header overheads, and at the same time, conserving the nice plug and play properties of the Ethernet MAC addressing. The iBF-based fine control over the path traveled by packets enables multiple load balancing schemes to avoid hot-spots, for instance, by bouncing off traffic flows to intermediate switches.

The rest of the paper is organized as follows. Section 2 introduces background information on the rationale behind rethinking DCN architectures and outlines highlights of related work. Section 3 presents the design principles adopted for our solution and describes the key functional blocks. In Section 4, we detail the prototype implementation and the testbed environment. Section 5 evaluates SiBF in terms of network state requirements, false positive performance, and load balancing capabilities. Finally, Section 6 concludes the paper and outlines the future work.

2. Background

Current efforts towards low-cost powerful computing facilities span from large-scale (geo)-distributed application programming, innovation in the DC infrastructure, and rethinking how to interconnect commodity PCs at large. Our work is focused on the latter. In this section, we first introduce networking requirements of the cloud, and then provide a snapshot of two remarkable new generation DCN proposals. Finally, we present the Bloom filter data structure, which is at the heart of our proposed forwarding mechanism.

2.1. Networking requirements of cloud data centers

The existing DCN literature seems to agree that efficiently networking the cloud DC calls for re-thinking the underpinning architecture to meet a reviewed set of requirements, which we have summarized as follows:

Resource Pooling: Offering the illusion of infinite computing resources available on demand requires means for elastic computing and agile networking. Such degree of DCN agility is possible (i) if IP addresses can be assigned to any VM within any physical server, and (ii) if all network paths are enabled and load-balanced.

Scalability: Networking (dynamically) a large pool of location-independent IP addresses (i.e., in the order of millions of VMs) requires a large scale Ethernet forwarding approach. Unfortunately, ARP broadcasts, MAC forwarding table sizes, and spanning tree limitations place a practical limit on the size of the system.

Performance: Available bandwidth should be high and independent from the endpoints' location, which requires congestion-free routing for any traffic matrix in addition to fault-tolerance (i.e., graceful degradation) to link and server instabilities.

Middlebox support: An ordered sequence of middlebox services (e.g., firewalls, WAN optimizers, load balancers) is commonly required to be (transparently) placed on the network paths of DCN traffic. Conventional solutions (e.g., SPT, VLAN, OSPF) turns

the overall configuration into a costly and tedious operation, besides unnecessary resource and performance inefficiencies [Joseph et al. 2008].

2.2. Related work

There is a large body of work tackling the cloud DCN research issues resulting in a collection of customized architectural proposals. We briefly outline the essence of two proposals which have inspired parts of our design.

PortLand proposes a scalable Ethernet-like layer 2 routing and forwarding protocol for data centers with three-tiered hierarchical topologies [Niranjan Mysore et al. 2009]. The approach to overcome the scalability limitations of Ethernet is based on modifying the control plane of the network, leaving the switch hardware and end hosts untouched. The main idea behind PortLand is the locator/identifier split, where nodes are identified by their actual MAC (AMAC) address, and located by a pseudo MAC (PMAC) address, which encodes hierarchical location information in its structure. Mapping between the two addressing spaces is done by the edge switches after querying a central fabric manager, which is responsible for tracking each correspondence of IP to pseudo MAC address within the discovered topology. Edge switches perform AMAC-PMAC rewriting for outgoing and incoming traffic.

VL2 provides a scalable Virtual Layer 2 to empower huge data centers with uniform high capacity between servers, performance isolation, and Ethernet semantics [Greenberg et al. 2009b]. Building upon existing technologies, in order to support agility, VL2 uses flat addresses in the IP layer to separate names from locators. VL2 yields uniform high capacity and traffic fairness by virtue of Valiant Load Balancing to randomize the traffic throughout the 3-tiered switching fabric using IP-in-IP encapsulation and Equal Cost Multi-Path (ECMP). Address resolution (i.e., application IP to location IP) is done modifying the end-systems and querying a scalable directory service.

2.3. Bloom filters

The Bloom filter is a popular data structure capable of answering questions of the form "is element x in set S?", with some tunable probability of returning false positives, i.e., claiming that x belongs to S even when this is not true. A typical implementation consists of a bit array of size m and k independent hash functions used to set/check bit positions when inserting/querying elements, which in our case are going to be switch MAC addresses forming a source route. The probability of false positives after inserting n elements is commonly approximated as (cf. [Bose et al. 2008]):

$$p^k = \left[1 - \left(1 - \frac{1}{m}\right)^{k*n}\right]^k \tag{1}$$

3. Design

The data center, as an interconnection network to perform distributed processing tasks, has three key dominant elements that determine its performance: (1) the network architecture (i.e. naming, address resolution, etc.), (2) the routing scheme, and (3) the interconnection topology. In this section we describe the design principles adopted to address (1) and (2) which can be summarized as an identifier/locator separated approach where IP

addresses act solely as identifiers and oblivious routing is provided by forwarding based on in-packet Bloom filters (iBF) encoding randomly selected routes between the communicating endpoints. As for (3), the interconnection topology, in line with the existing literature, we assume a 3-tier topology with a lower layer of ToR switches, an intermediate layer of p_1-port Aggregation (AGGR) switches, and an upper layer of p_2-port CORE switches (see Fig. 1). Our solution is not restricted to a particular topology, and works on e.g., 3-level fat-trees with identical p-port switches (like Portland) or 3-tier 5-stage Clos arrangements (with $p_1 \neq p_2$ like VL2). Moreover, we note that other scale-out topologies could be considered (e.g., DHT-like rings, Hypercubes, Torus, etc.), as long as they offer large path diversity and low diameter.

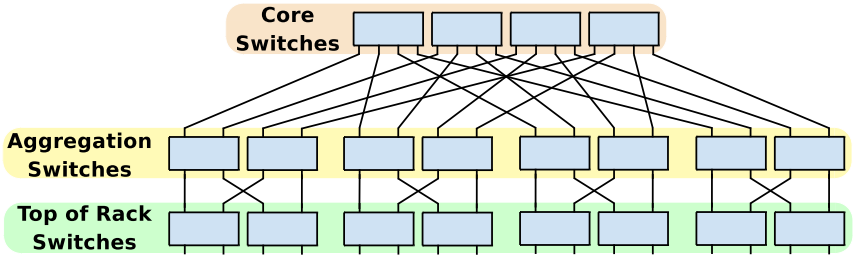


Figure 1. A 3-tier fat tree using 4-port switches.

3.1. Design Principles

We adopted the following principles for the proposed data center network architecture:

Separating Names from Locations: Identifier-locator split is the fundamental capability to enable resource pooling of IP addressable services, which can expand or contract their footprint in the DC as required (*agility*). IP addresses are used to identify physical servers (and VMs) within the DC. That is, no topological constraints are imposed on how IP addresses are assigned or translated in case of communications towards external networks (i.e., public Internet). In this context, IP addresses are not meaningful for packet routing, which is solely based on a revisited source-routing capable Ethernet layer.

Source explicit routing with zero-overhead: Leveraging the small diameter of data center topologies, our approach to meet the scalability goals is based on *strict source routing*. Routing in 3-tier DCN topologies is fairly simple, as any route between two ToRs, has an upward phase towards a common CORE switch and then a downward path to one AGGR switch connected to the destination ToR. Forwarding is based on an iBF containing only three elements, namely the Bloomed MAC identifiers of $\langle CORE_i, AGGR_{down}, ToR_{dst} \rangle$ switches. Source ToRs encode the iBFs in the MAC fields of outgoing packets which are sent to a next hop $AGGR_{up}$ switch. Hence, three iBF-based forwarding decisions are taken, one at the first AGGR, one at the CORE and one at the down-path AGGR. The destination ToR needs to re-write the source and destination MAC fields before delivering the packet to the destination server. By carrying the iBF in the 96 bit space of the MAC fields and re-writing packets at ToRs, we avoid encapsulation techniques or additional shim headers. Source routing not only minimizes FIB requirements of intermediate switches but also eases the inclusion of middleboxes.

Direct network control and logically centralized directory: SiBF embraces the 4D [Greenberg et al. 2005] philosophy of simplifying the data plane and centralizing the

control plane to enforce the data center goals. We introduce the role of a Rack Manager (RM) to take the routing decisions and program the state of programmable switches. In order to construct source routes, two pre-requisites are required: (1) *topology information*, and (2) *server location*. We surmise that a directory service to track host locations and the underlying switching topology are implementable and able to scale to the envisioned DCN demands as shown by related work (e.g., Tesseract, Bing's Autopilot, VL2).

Load Balancing through path randomization: The approach to provide load balancing is based on *oblivious routing*, i.e., traffic independent randomized packet routing [Yuan et al. 2007]. More specifically, like VL2, we implement valiant load balancing (VLB) using the routing iBFs to "bounce off" flows at random intermediate switches.

Unmodified endpoints and plug-and-play: The forwarding fabric does not rely on end-host modifications. Legacy servers, operating systems and applications are supported off-the-shelf. Moreover, the plug-and-play behaviour of Ethernet is to be conserved, with auto-configuration of end-hosts and switches being part of the solution.

3.2. False-positive-free forwarding on Bloomed MAC identifiers

The key innovation comes when "switching" in the AGGR and CORE layers. Forwarding tables of switches are initially empty and get filled with one flow entry per neighboring switch detected. A flow entry is wildcarded except for the 96 bits of the source and destination MAC fields. However, instead of traditional exact matching of MAC fields, each flow entry contains a 96-bit mask generated from k hashes of the neighbouring switch unique MAC address. Similar to Link IDs in [Jokela et al. 2009], a *Bloomed MAC ID* is a 96-bit vector where only k bits set, but with the key difference that it is *not* directional, i.e., generated on a network interface pair basis. Forwarding decisions are trivial. On packet arrival, only the k 1s of each Bloomed MAC ID are checked for presence in the Ethernet MAC fields carrying the iBF. Upon match, the packet is forwarded.

Generation of Bloomed MAC identifiers: Instead of making k independent hashes of the neighboring switch MAC address, we make only one hash using a cryptographic function (e.g., MD5) and concatenate the output with the least significant 24-bits of the MAC address (unique per Ethernet vendor). Thereby, we obtain a randomly generated $128 + 24$ bit vector, which we slice in 7-bit segments to obtain k "pseudo" hashes that determine the bit positions in the 96-bit Bloom filter:

$$iBF[i] = (MAC_{24:48}|MD5(MAC))[7i : 7(i+1)] mod 96 \quad (2)$$

Bloomed MACs IDs generated this way are still statistically unique e.g., $m!/(m-k)! \approx 10^{13}$ for $m = 96$ and $k = 10$. The algorithm defined by Eq. 2 is a system wide parameter that can be changed or optimized for a given set of MAC addresses (if required).

False positives: The well-known caveat of Bloom filters is the possibility of returning false positives to set membership queries, i.e., returning true when a set element was not inserted. In our case, this means that in addition to the explicitly inserted next hop switch, additional switch(es) appear(s) as next hop candidate(s). The resulting conflict can be solved either (i) by multi-casting the packet along all matching interfaces, or (ii) by picking only one. In any case, we require *iBF forwarding completeness*, i.e., loop-free and guaranteed delivery of packets to the intended destination(s). After a careful analysis of every false positive case of our implementation choice, we claim to have

a loop-free, high solution that circumvents any potential issue arising from false positives. The factors that contribute to this result are multi-fold, some of them are due to an iBF-forwarding design tailored for multi-rooted tree topologies, and the remaining are implementation-specific, i.e. forced/enabled by our OpenFlow implementation choice. To start with, note that due to the high bit per element ratio ($m/n \approx 30$ for $m = 96$ and $n = 3$), false positives are extremely rare i.e. in the order of 10^{-7} (see details in Sec. 5.2).

Our strategy to avoid the potential effects of false positives is to exploit the notion of *power of choices* along two dimensions: (1) multiple paths, and (2) multiple iBF representations. That is, we compute the iBFs of the multiple available paths, and for each we generate d additional candidates using different sets of hash functions. Using the topology information, the routing service can easily check *a priori* whether any candidate iBFs is prone to false positives along the path. If so, those candidate iBFs are discarded from the random path selection. For the sake of brevity, we omit some details of the OpenFlow implementation and the analysis of why some false positives are self-healed by virtue of the multi-rooted topology. In a nutshell, false positives result in multiple flow entries matching the wildcarded k bits in the iBF. Since only the actions associated to one entry can be executed (per OpenFlow specification), a packet may be wrongly forwarded to a switch not included in the source route. Such packets lacking of matching flow entries are forwarded to the RM, which computes an alternative path and installs the required flow entries to temporarily fix the issue. However, recall that our strategy to avoid false positives is to detect them *prior* to their use. With knowledge of the topology, the RM pre-computes and maintains a source to destination ToR matrix filled only with false-positive-free iBFs for the multiple available network paths. In Sec. 5.3 we experimentally quantify the penalties on path multiplicity, which we anticipate to be insignificant.

3.3. Tree and Role Discovery Protocol

Topology knowledge is a prerequisite to allow source routing. A point which is not so evident and trivial is how to correctly infer the tree topology and the role of each switch (i.e., ToR, CORE or AGGR.), more critically at bootstrap time, since one of our requirements is to mimic the Ethernet plug & play behavior to avoid any manual intervention. This feature does not only reduce operational efforts to e.g., replace misbehaving switches, but is critical for the correct (and optimized) routing of packets. To this end, we have designed a Role Discovery Protocol (see Algorithm 1) that automates the inference of the switching tree by simply extending the link layer discovery protocol (LLDP) with an extension TLV to include the discovered role. We note that Portland faced a similar challenge in order to switches discovering their specific location within the DCN hierarchy to form a pseudo MAC address of the form *pod.position.port.vmid*. Our protocol is fairly simpler and requires only to identify the layer in which it is located.

4. Prototype implementation and testbed

Implementation of the iBF-forwarding mechanism is based on OpenFlow switches [McKeown et al. 2008], and the Rack Manager (RM) has been implemented as an application on top of the NOX controller [Gude et al. 2008]. In the following, we describe the key issues of the implementation work and the testbed environment. For details on the prototype implementation and on how to replicate our testbed we refer to

Algorithm 1: Role Discovery Protocol.

```
begin switch_join
    ROLE ← UNDEFINED;
    SendAllPorts(llpd, ROLE);
end
begin arp_receive_server
    if ROLE != TOR then
        ROLE ← TOR;
    end
end
begin lldp_receive_neighbors
    NBROLE ← neighbors.ROLE;
    if NBROLE = (CORE or TOR) then
        ROLE ← AGGR;
    else if NBROLE = AGGR then
        ROLE ← CORE;
    end
end
```

the publicly available source files and how-to instructions.[1]

4.1. OpenFlow

An OpenFlow (OF) switch separates the fast packet forwarding (data path) from the high level routing decisions (control path) of a router or switch. While the data path portion still resides on the switch and runs using the same underlying hardware, high-level packet handling decisions (i.e. routing) are moved to a separate controller. OF-enabled devices and the controller(s) communicate via the OF protocol, which defines messages, such as `packet-received`, `send-packet-out`, `modify-forwarding-table`, and `get-stats`.

The disruptive aspect of OF is to define a clean interface in form of a flow table abstraction with entries containing a set of packet fields to match from the 10-tuple: $\langle inport, Eth_{src}, Eth_{dst}, VLAN, EthType, IP_{proto}, IP_{src}, IP_{dst}, TCP_{src}, TCP_{dst} \rangle$, and a list of hardware-supported actions, i.e., `send-out-port`, `modify-field`, or `drop`. When an OF switch receives a packet for which it has no matching flow entry, it sends this packet to the controller, which in turn decides on how to handle the packet. The decision is sent to the switch, which can be instructed to cache the decision for some period of time by adding a flow entry to handle upcoming packets at line rate.

In order to support the iBF-based forwarding, only a minor modification was required to the current OpenFlow reference implementations (v.0.89rev2 and v.1.0). The key of iBF-based forwarding is the Bloomed MAC identifier which is a wildcarded *bitmask* with only k arbitrary bits set to one. Thus, we needed to add this special behavior support to the OpenFlow datapath implementation. Fortunately, this required only changes in two lines of code[2] of the fast path flow matching function.

[1] http://www.dca.fee.unicamp.br/ chesteve/

[2] function `flow_fields_match` in `openflow1.0.0/udatapath/switchflow.c` or `openflow0.9.0/datapath/flow.c`

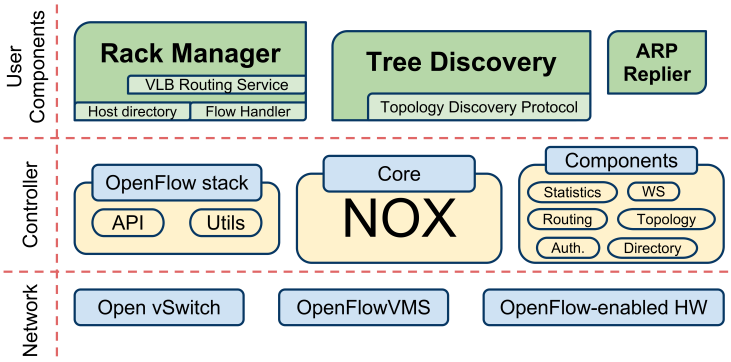


Figure 2. Component Architecture.

4.2. Rack Manager

The Rack Manager acts as a controller of OF switches, and as such, the natural implementation is as an application on top of the open source OF controller named NOX [Gude et al. 2008]. In a nutshell, NOX's programmatic interface is built upon *events*, triggered by NOX core components, thrown by user-defined applications, and generated directly from OF messages like `packet-in`, `switch join`, `switch leave`, etc. Figure 2 depicts our implementation of the RM functionality, which we have divided into three separate NOX user components.

4.3. Message sequence

The packet flow diagram of Fig. 3 shows how communications happen in the prototype implementation. Regular arrows are single data packets and the dotted arrows represent OF protocol messages. A server's network activity starts by sending an ARP request to some destination IP_x (Step 0), for instance, to resolve the address of the DNS server. The ARP request reaches the ToR, which has no matching entry and informs the controller.

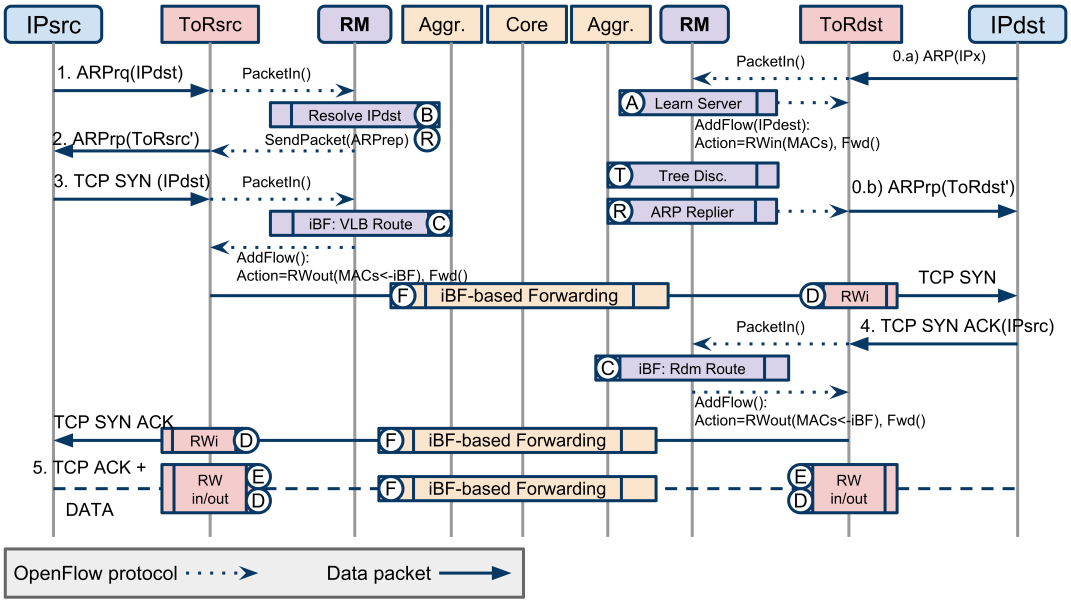


Figure 3. Packet flow sequence in an OpenFlow-based SiBF instantiation.

The *packet-in* event is passed to all the modules which have expressed interest in ARP packets. The RM learns the server location and registers it as being attached to a port of the OF switch triggering the event (Action A). It then sends a *flow-mod* command to install a semi-permanent flow entry for future incoming packets containing the destination IP equal to the server IP, and the associated actions set to (i) re-write the MAC fields with the ToR and server original MAC addresses, and (ii) forward to the attached port. The Tree Discovery identifies the switch as a ToR and updates the state of the Role Discovery Protocol (Action T). The ARP replier responds with a "fake" ARP reply containing the ToR MAC (Action R). In Step 1, a server sends an ARP request for a destination IP, and like any originating ARP, it acts as a trigger for the server discovery actions described in Step 0. After receiving the ARP reply (Step 2), the source node sends a TCP SYN packet which hits the ToR switch and is accordingly forwarded to the controller (Step 3). The RM picks one iBF towards the destination ToR (Action C), and orders the installation of an OF entry (10 sec. soft-expiration) to re-write packets belonging to the fully specified 10-tuple flow. Packets within this flow description get the iBF written in the MAC fields and are forwarded at line rate across the AGGR and CORE layers based on the iBF source route (Action F). When the iBF-labeled packet hits the destination ToR, it matches the flow entry installed in Step 0 (Action A) and is delivered to the destination server after re-writing the MAC headers (Action D). In Step 4, the destination server replies with a TCP SYN ACK which lacks of a flow entry and is delivered to the RM (Action C). After iBF selection and the installation of the flow entry (Action C), the TCP SYN ACK is forwarded based on the iBF. Upon reception at the originating server, the 3-way handshake can be completed (Step 5) and both entities can exchange data at line rate.

4.4. Testbed

The testbed consists of 5 physical nodes, one hosting the NOX controller with the RM components and the remaining 4 were partitioned into 9 virtual machines: 5 instantiating an OF switch each, and 4 hosting linux-based VMs. Figure 4 shows the testbed environment, where the solid lines represent direct links between virtual machines and the dashed lines represent the connections between VMs from different physical machines. The topology on each physical machine is configured with OpenFlowVMS, which in-

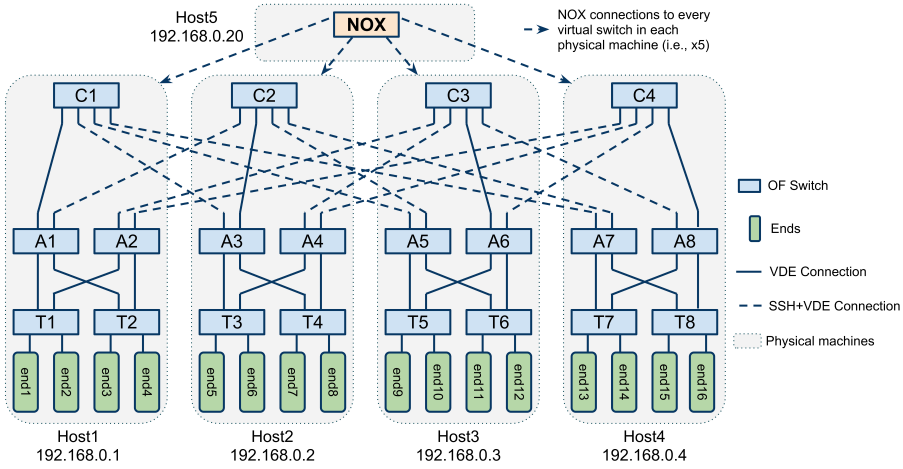


Figure 4. Testbed environment.

Table 1. Evaluation of the state requirements in terms of entries at switches.

Physical hosts	2.880			23.040			103.608		
Racks	144			1152			5184		
Aggr. Switches	24 ($p_1 = 24$)			96 ($p_1 = 48$)			144 ($p_1 = 144$)		
Core Switches	12 ($p_2 = 24$)			24 ($p_2 = 96$)			72 ($p_2 = 144$)		
	VL2	Portland	SiBF	VL2	Portland	SiBF	VL2	Portland	SiBF
Entries at ToR	200	120	120	1292	120	120	5420	120	120
Entries at AGGR	180	24	24	1272	48	48	5400	144	144
Entries at CORE	180	24	24	1272	96	96	5400	144	144

cludes a useful set of scripts to automate the creation of networked VMs using QEMU and VDE. Additional scripts were developed to distribute the environment across different physical machines using *ssh* connections and virtual dumb-switches based on VDE. Our extended script set enables to quickly define a target topology and automate the bootstrapping of the virtual nodes and OF switches, including the IP configuration, the creation of data-paths, the start-up of OF modules and the connection to the controller.

5. Evaluation

After validating the prototype implementation by verifying the full connectivity among the pool of servers (16 VMs), the next question is to evaluate the iBF-based forwarding fabric in terms of (i) state requirements, (ii) potential effects of false positives, and (iii) the load balancing capabilities. Due to the limitations of a virtualized testbed, performance aspects like goodput and flow completion times are left out of scope here.

5.1. State analysis

We start by comparing analytically the state requirements of SiBF with VL2 and Portland. The network setup is a 3-tier Clos topology, with ToRs connecting to 20 servers via 1 Gbps ports and to two AGGRs via 10 Gbps links. The p_1 ports of AGGRs are used to connect to $p_1/2$ ToRs and $p_1/2$ COREs equipped with p_2 high speed ports. In line with related work [Tavakoli et al. 2009], we assume an average of 10 concurrent flows per server (5 in and 5 out). Table 1 presents the scalability requirements for different switch configurations. By virtue of strict source routing, SiBF requires minimal state at COREs and AGGRs, namely only one entry per interfacing neighbor. Moreover, scaling-out the DCN does not impact the number of flow entries in the switches which is constant and equal to the number of neighbors. At ToRs, the amount of flow entries grows with the number of concurrent outgoing flows plus a constant amount of entries, one for each hosted server in order to re-write terminating flows. By comparison, VL2 requires forwarding entries in proportion to the total number of switches in order to route packets along the two-levels of IP encapsulation: $\langle LA_{CORE}, LA_{ToR} \rangle$. On the other hand, Portland has the same state requirements as SiBF, namely only one forwarding entry per interface, sufficient to perform the hierarchical forwarding on PMACs.

5.2. False positives

Now, we turn our attention to the practical false positive performance of small 96-bit Bloom filters when holding only 3 elements, namely the three Bloomed MAC addresses. More than the theoretical estimates (i.e., Eq. 1), what practitioners are really interested is in the observed false positive rate (fpr) after the iBF is queried for elements. Therefore,

Table 2. Evaluation of the false positive rate of the 96-bit iBF.

k	5	6	7	8	9	10	11	13	15	17	19	21
Theor. Eq 1 ($\cdot 10^{-6}$)	64.89	25.7	11.68	5.95	3.33	2.03	1.32	0.68	0.42	0.31	0.25	0.23
fpr ($\cdot 10^{-4}$)	2.41	1.81	1.5	1.7	1.83	2.23	3.09	4.92	7.17	11.46	16.09	21.07
fpr_{min} ($\cdot 10^{-6}$)	0.93	0.58	1.74	1.85	2.78	5.56	9.72	28.6	95.1	182	355	591

from a pool of 1M unique, randomly generated 48-bit values, on each experiment round (10.000 in total) we randomly insert 3 of them into a 96-bit BF using the Bloom MAC ID algorithm (Eq. 2) and test for presence of 432 (= 144 * 3 hops) randomly selected MACs. Table 2 shows the observed fpr for basic BFs constructs and when the power of choice optimization (with $d = 4$, $m' = 94$) is used (fpr_{min}). In theory, the optimal number of hash functions ($k_{opt} = \frac{m}{n} ln2$) that minimizes the false positive probability would be as many as 22. However, in our practical setup, the lowest fpr was obtained for k around 7. The deviation from the theoretical estimates can be explained by the accurate equation and bounds for small size Bloom filters by Bose et al. [Bose et al. 2008, Theorem 3]. Even without the d-candidate extension, only a few false positives per 10.000 queries were observed in plain 96-bit iBFs, which suggests that the effect of a false positive (if any) could be easily handled on a per-case basis.

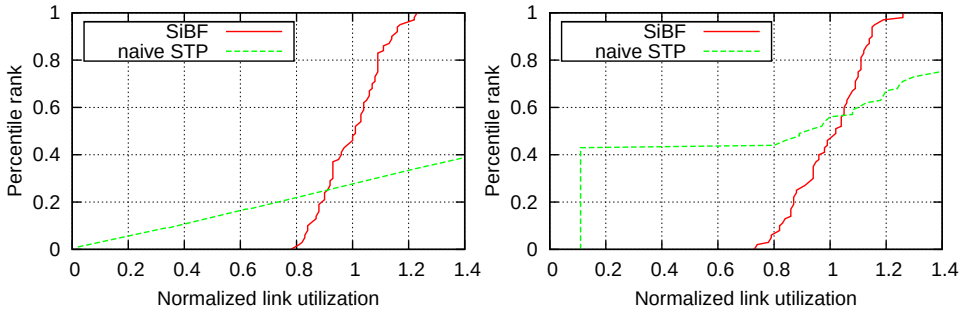
5.3. False-positive-free forwarding on large-scale DCN topologies

We now evaluate the viability and efficiency of our false positive avoidance strategy based on discarding false-positive-prone iBF candidates prior to their use. Our thesis is that, given the low fpr of the 96-bit iBF data structure, there are plenty of false-positive-free paths between any two communicating nodes. In this experiment, we use an ns-3 implementation to explore the fpr performance on large-scale DCN topologies by sending an iBF for each of the available path between every ToR. Following the approach described in Sec. 5.1, we generate a topology with 48-port AGGRs and COREs to interconnect 576 ToRs, enough to host 11.520 physical servers. Testing every combination of $\langle ToR_{src} - ToR_{dst} \rangle$ (i.e., 331.200 ToR pairs) along each available path, results in over 30M iBFs sent and accounted for false positives. The summary results are as follows: 74% of the ToR pairs were false-positive-free for every available shortest path. Among those with some false positive (26%), the average number was 3 out of the 96 multiple paths. The maximum number of false positive paths for any ToR combination was 10. As a result, only 0.92% of all network paths exhibited some false positive and should be kept out of the pool of iBFs used for load balanced routing. Based on these results, we may conclude that false-positive-free forwarding comes at an affordable cost (less than 1%) in reduced path multiplicity. Moreover, considering the d-candidate optimization with e.g., $d = 4$, we could, with very high probability, get rid of the remaining 1% of false-positive iBFs by choosing alternative bit representations, and thereby utilize every available path.

5.4. Load balancing capabilities

Now, we investigate the load balancing capabilities of implementing VLB with iBFs over our testbed environment. Given a traffic matrix (TM), the goal is to evaluate how well the traffic is spread among the available links. We compare the link utilization of our VLB implementation with a vanilla Spanning Tree (SPT) implementation over the same topology. Two types of TMs were tested, one to mimic the all-to-all characteristics of DC

applications like MapReduce, and one with random communicating endpoints. We used ITG [Avallone et al. 2004] as the traffic generator configured with TCP flows to last for 10s, with exponentially distributed payload sizes around 850 Bytes, which are reasonable assumptions for the majority of the reported DCN traffic. Figure 5 shows the normalized link utilization after ten experiment runs. As expected, SPT under- and over-utilizes the network links, whereas SiBF spreads traffic remarkably well, with the maximum and minimum normalized utilization of any link deviating only around 20% from the ideal value, i.e., 1. In the case of randomly chosen endpoints (Fig. 5(b)), the conclusion is the same, VLB using iBFs achieves a nice utilization of the available links in a TM-independent manner. The distribution of the normalized link utilization is comparable to the numbers reported in the VLB implementation of VL2, with min values (0.78 vs. 0.46) and max values (1.23 vs 1.2) [Greenberg et al. 2009b, Fig. 15]. The divergence of the min values can be explained by the nature of the operational traffic in VL2 compared to our synthetic TMs.



(a) All to all. (b) Random.

Figure 5. Evaluation of the load balancing behaviour. CDFs of the link utilization.

6. Conclusion

We have presented SiBF, a data center network architecture based on a simple data plane layer below IP that forwards packets based on the contents of an in-packet Bloom filter. SiBF embraces the (upcoming) category of commodity switches leveraged with a flow-oriented API extending the next frontier in data center networks from "commoditization" to "customization." The DCN proposal presents many appealing characteristics such as not requiring any modification of end-hosts, reusing the Ethernet packet header bit space, minimal FiB consumption, and a fine control over the packet routes across the data center. The evaluation on a small-scale virtualized testbed implementation not only provides a proof of concept that helped to feedback the design cycles, but also shed light on the actual capacity of providing load balancing with randomized iBFs. In future implementation rounds, the prototype will be improved (e.g., to handle failure cases) and extended with additional features like distributed database management (e.g., topology and host directory) and transparent middlebox traversal, making it all together a real candidate to be deployed as an in-house cloud DCN playground.

Acknowledgements

This work is partly funded by CNPq, Capes, FAPESP and Ericsson Research.

References

Al-Fares, M., Loukissas, A., and Vahdat, A. (2008). A scalable, commodity data center network architecture. *SIGCOMM CCR*, 38(4):63–74.

Avallone, S., Guadagno, S., Emma, D., Pescape, A., and Ventre, G. (2004). D-itg distributed internet traffic generator. In *QEST '04*. IEEE Computer Society.

Benson, T., Anand, A., Akella, A., and Zhang, M. (2009). Understanding data center traffic characteristics. In *WREN '09*. ACM.

Bose, P., Guo, H., Kranakis, E., Maheshwari, A., Morin, P., Morrison, J., Smid, M., and Tang, Y. (2008). On the false-positive rate of Bloom filters. *Information Processing Letters*, 108(4):210–213.

Greenberg, A., Hamilton, J., Maltz, D. A., and Patel, P. (2009a). The cost of a cloud: research problems in data center networks. *SIGCOMM CCR*, 39(1).

Greenberg, A., Hamilton, J. R., Jain, N., Kandula, S., Kim, C., Lahiri, P., Maltz, D. A., Patel, P., and Sengupta, S. (2009b). VL2: a scalable and flexible data center network. *SIGCOMM CCR*, 39(4):51–62.

Greenberg, A., Hjalmtysson, G., Maltz, D. A., Myers, A., Rexford, J., Xie, G., Yan, H., Zhan, J., and Zhang, H. (2005). A clean slate 4D approach to network control and management. *SIGCOMM CCR*, 35(5):41–54.

Gude, N., Koponen, T., Pettit, J., Pfaff, B., Casado, M., McKeown, N., and Shenker, S. (2008). NOX: towards an operating system for networks. *SIGCOMM CCR*, 38(3).

Guo, C., Lu, G., Li, D., Wu, H., Zhang, X., Shi, Y., Tian, C., Zhang, Y., and Lu, S. (2009). Bcube: a high performance, server-centric network architecture for modular data centers. In *SIGCOMM '09*. ACM.

Jokela, P., Zahemszky, A., Esteve Rothenberg, C., Arianfar, S., and Nikander, P. (2009). LIPSIN: line speed publish/subscribe inter-networking. In *SIGCOMM '09*. ACM.

Joseph, D. A., Tavakoli, A., and Stoica, I. (2008). A policy-aware switching layer for data centers. *SIGCOMM CCR*, 38(4):51–62.

McKeown, N., Anderson, T., Balakrishnan, H., Parulkar, G., Peterson, L., Rexford, J., Shenker, S., and Turner, J. (2008). OpenFlow: enabling innovation in campus networks. *SIGCOMM CCR*, 38(2):69–74.

Niranjan Mysore, R., Pamboris, A., Farrington, N., Huang, N., Miri, P., Radhakrishnan, S., Subramanya, V., and Vahdat, A. (2009). Portland: a scalable fault-tolerant layer 2 data center network fabric. In *SIGCOMM '09*. ACM.

S. Kandula, Sudipta Sengupta, A. G. and Patel, P. (2009). The nature of data center traffic: Measurements and analysis. In *ACM SIGCOMM IMC*.

Tavakoli, A., Casado, M., Koponen, T., and Shenker, S. (2009). Applying NOX to the datacenter. In *Proc. of workshop on Hot Topics in Networks (HotNets-VIII)*.

Yuan, X., Nienaber, W., Duan, Z., and Melhem, R. (2007). Oblivious routing for fat-tree based system area networks with uncertain traffic demands. *SIGMETRICS PER*, 35(1).