Received April 17, 2021, accepted April 26, 2021, date of publication May 6, 2021, date of current version May 14, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3077955

Stateless Flow-Zone Switching Using Software-Defined Addressing

SERGIO GONZALEZ-DIAZ[1], **ROGER MARKS**[2], **ELISA ROJAS**[3], **ANTONIO DE LA OLIVA**[4], **(Member, IEEE), AND ROBERT GAZDA**[5]
[1]Atos Spain, 28037 Madrid, Spain
[2]EthAirNet Associates, Denver, CO 80207, USA
[3]Departamento de Automática, Universidad de Alcalá, 28805 Alcalá de Henares, Spain
[4]Department of Telematic Engineering, University Carlos III of Madrid, 28911 Leganés, Spain
[5]InterDigital Communications, Conshohocken, PA 19428, USA

Corresponding author: Sergio Gonzalez-Diaz (sergonzalezdiaz@gmail.com)

This work was supported in part by the H2020 Europe/Taiwan Joint Action 5G-DIVE under Grant 859881, in part by the Spanish State Research Agency through the TRUE5G Project under Grant PID2019-108713RB-C52/AEI/10.13039/501100011033, and in part by the Comunidad de Madrid through the Project TAPIR-CM under Grant S2018/TCS-4496.

ABSTRACT The trend toward cloudification of communication networks and services, with user data and applications stored and processed in data centers, pushes the limits of current Data Center Networks (DCNs), requiring improved scalability, resiliency, and performance. Here we consider a DCN forwarding approach based on software-defined addressing (SDA), which embeds semantics in the Medium Access Control (MAC) address and thereby enables new forwarding processes. This work presents Flow-Zone Switching (FZS), a loop-free location-based source-routing solution that eliminates the need for forwarding tables by embedding routing instructions and flow identifiers directly in the flow-zone software-defined address. FZS speeds the forwarding process, increasing the throughput and reducing the latency of QoS-sensitive flows while reducing the capital and operational costs of switching. This paper presents details of FZS and a performance evaluation within a complete DCN.

INDEX TERMS Flow-zone switching (FZS), data center network (DCN), software-defined addressing (SDA), layer 2, MAC address, routing, forwarding, QoS, clos, stateless.

I. INTRODUCTION

Data center demand has increased profoundly in the last decade, driven largely by the success of the cloud computing paradigm. Data center scales have grown from hundreds of servers to thousands; while 5000 servers is typically considered the minimum for a "hyperscale" data center [1], the numbers grow much larger. Meanwhile, the same application-driven trends have put additional emphasis on the performance and cost of the Data Center Network (DCN) that provides a critical role in data center infrastructure. DCN designs (e.g. [2]) aim to provide responsive services to clients based on real-time extraction of results from vast stored resources. In the background, these data centers simultaneously analyze and systematically extract massive amounts of information from extensive data sets, typically using artificial intelligence algorithms, with data that is dispersed across the data center. In such a scenario, efficiency, scalability, and resiliency are three key pillars supporting increased performance. Capital cost and operating costs, notably those due to energy consumption, are also critical.

In recent years, Software-Defined Networking (SDN) has played a major role in the management and orchestration of the DCNs, increasing the flexibility and programmability by logically centralizing the network control plane. As in legacy networks with more distributed control, SDN data plane forwarding is based on forwarding tables. These forwarding tables are filled with a huge number of table entries, typically specifying actions based on Medium Access Control (MAC) addresses (in Layer 2 DCNs) or Internet Protocol (IP) addresses (in Layer 3 DCNs), or combinations thereof. Since the DCN topology may accommodate many thousands of servers, each running many Virtual Machines (VMs) and applications, forwarding tables may be huge, imposing significant local memory requirements at the switches.

The associate editor coordinating the review of this manuscript and approving it for publication was Ting Wang.

Serious scalability problems arise, due to an increasing number of switches and table entries therein, including issues related to network management complications. These forwarding table problems are particularly troublesome for Layer 2 networks. While IP addresses are structured so as to convey network information, MAC addresses are typically considered to be flat. Aside from two meaningful bits in the 48-bit MAC address, the remainder of the MAC address has been considered to serve no purpose other than to provide uniqueness. However, following trends in standardization, we have considered the possibilities of flexible Software-Defined Addressing (SDA) and how it can be exploited for improved network performance and efficiency. Like some prior approaches, we embed routing instructions semantically into MAC addresses, with the destination address explicitly identifying the topological zone of the recipient with sufficient instructions to route a frame most directly to that destination, eliminating the need for forwarding tables entirely and resolving Layer 2 concerns such as address learning and looping. The flexibility provided by SDA, applied to our FZS architecture, allows the implementation of diverse routing and traffic engineering schema such as source routing or elephant/mouse differentiation among others, with the only constraint of requiring a quasi-hierarchical network to operate. Please note that source routing (and segment routing specifically) is just one of the possible mechanisms that can be implemented using FZS and SDA.

In our examples, we segment the DCN topology in four zone levels, representing the server level at the bottom and the spine level at the top, passing through the rack level (with Top-of-Rack (ToR) switches) and the pod level. We assign each zone an ID in a quasi-hierarchical manner: each pod has an identifier, each rack has a local identifier within its pod, and each server has a local identifier within its rack. Each server is therefore identified with a unique zone in the topology, identified by the pod, rack and server identifiers. Spines are uniquely identified but are not associated with server zones.

Here we enhance prior approaches by embedding specific flow identification into the address as well. As a result, the address differentiates and explicitly identifies the various flows directed toward a particular destination. This explicit flow identification can be exploited in many ways, such as ensuring that all frames in a flow follow the same route, differentiating the frames of a particular server by identifying the VM, differentiating the frame QoS by flow, and allowing header suppression per flow. Flow identifiers may be used to classify frames according to categories that may not normally be considered "flow" categories. We refer to the method as Flow-Zone Switching (FZS) because the frame codes both the zone identifier and flow identifiers. FZS minimizes the state in switches and, by embedding the routing instructions directly in the MAC addresses, can completely remove the need for forwarding tables.

To illustrate the potential use of FZS, we focus here on examples in which the flow identifier is used to distinguish latency-sensitive small-frame (mice) flows from large data-intensive (elephant) flows, as described elsewhere [3]. We arrange to route the mice and elephant flows differently, segregating the mice in low-delay paths that are source-coded into the frames. We show how this can translate into better performance and lower network congestion.

Data center source-routing solutions based on zone addressing have been already explored in the literature. The majority of existing solutions are based on address translation, reducing but not completely removing the need for table lookups. FZS aims to enhance the state of the art by completely removing the lookup tables and at the same time providing fine granularity for specific flow management, boosting the DCN performance while reducing its cost.

Unlike DCN routing methods based on overlays, FZS routing is conducted end-to-end at Layer 2, with routing instructions embedded in the MAC address. No overlay headers, additional frame tags, or other performance-reducing frame overhead is introduced. FZS offers the potential for a high performance with low complexity.

The following are the key contributions of this work:

- The routing mechanism minimizes the state in switches by embedding the routing instructions directly in the MAC addresses, completely removing the need for forwarding tables.
- Flow identifiers, embedded in the MAC address, are assigned directly at the frame source, allowing a fine-grained distinction of flows, rather than a course-grained version based on later frame classification. For example, TCP data segments can be classified as elephants and TCP ACKs as mice, even though they contain identical TCP/IP five-tuples. This allows new traffic engineering possibilities.
- Flow identifiers are exposed to all switches, allowing diverse flow management methods, as well as to the receiving end station, for post-reception processing such as header decompression indexed by the fine-grained flow.
- Each frame contains the path towards its source, as well as to the destination, with separate source-based and destination-based flow fields, without the intervention of any other protocol or mechanism. This means, for example, that the destination can directly learn the Layer 2 flow-zone address associated with the IP address of a received frame and can respond to that frame without the need to consult an external ARP server.
- No overhead is introduced into the frame, since all identification is included in Layer 2 headers that are unavoidable in any Ethernet transmission. No overlay headers or labels are used.
- No distribution of control (e.g., label to path binding) is required; all control is managed by the hosts.
- Using a 48-bit MAC address, the method is scalable to many orders of magnitude larger than current data center and can be scaled larger if necessary.

- Thanks to its programmability and flexibility, FZS can implement arbitrary traffic engineering mechanisms, even separating packets that, per traditional 5-tuple characterization, belong to the same flow (e.g., TCP data segment and TCP ACK).

Additional details regarding the benefits of FZS are provided in Section VII.

The rest of the paper is organized as follows. Section II provides a brief background. Section III summarizes related work. Section IV explains the concept of Software-Defined Addressing (SDA) and semantic MAC addresses in the context of recent standardization. In Section V, we detail the FZS structure and architecture, including zones, flows, and the address formats. Section VI details the forwarding operations, the stateless configuration, and an address assignment procedure. Section VII summarizes potential benefits of FZS. Section VIII focuses on the implementation details of a validation study compared to a table-based legacy solution. Section IX presents a performance evaluation of FZS in a DCNs using several Transmission Control Protocol (TCP) algorithms and traffic distribution solutions, comparing it to a legacy solution. Section X suggests possible areas for followup research. Finally, Section XI draws the conclusions of the paper.

II. BACKGROUND

Data center architectures are based on multi-tier topologies, typically structured as a generalized fat-tree [4]. The generalized fat-tree network is based on a folded Clos network topology and is usually constructed with many commodity switches instead of complex ones. These topologies provide multiple paths between pairs of communicating nodes. This allows networks to scale in bandwidth by adding ports and switches rather than simply by scaling the bandwidth of the physical link. This results in better cost/performance ratio as well as increased resilience with respect to link failure. The existence of multiple paths leads to the well-known problem of loops. Many technologies have been developed to overcome the loop problem in Layer 2 networks, but many of these, such as spanning tree algorithms, are based on limiting the available paths and therefore counteract the intention to provide multiple parallel routes for bandwidth purposes. Traditional Layer 2 networks also face scaling problems due to the flat network address that requires a forwarding table entry for each endpoint. Layer 2 approaches to this scaling issue, such as provider bridging and provider backbone bridging, have been introduced but are not popular in DCNs, possibly due to their added complexity and frame overhead due to the extra overlay encapsulation. These issues have kept Layer 2 networking from widespread use as a basis of routing in the massive data center.

Recently, a number of routing methods have been introduced for generalized fat-tree topologies. Many of these are based on a form of source routing, in which the routing instructions are entirely or partially embedded in the packet or frame. In some cases, the information is embedded in an IP address, a MAC address, or a customized label. FZS follows this general approach.

III. RELATED WORK

A flow-zone destination address conveys a zone identifier, which specifies a location and therefore a portion of the path to the destination, as well as flow information, which can be configured to specify the remaining portion of the path starting at the source. Thus, FZS is a form of source routing (SR). Layer 2 source addresses conveying routing information alone have been explored previously in the literature. Segment routing is a form of source routing, recently standardized in the IETF Source Packet Routing in Networking (SPRING) Working Group,[1] in which routing is specified along segments that comprise the entire path. While FZS does not accord with the SPRING standard, it can be viewed as a form of segment routing, particular when addresses route the frame specifically only over a portion of the path.

Methods pertaining to our research can be classified based on how the routes are carried in the packets:

- Label-based SR: Methods in this category append labels to packets or frames indicating the path to be followed towards the destination. MPLS is typically used to encapsulate the labels, therefore achieving SR in legacy MPLS networks. Work in this area includes [10], [28]–[30], [33], [37], [38], [44] and [23].
- Address-based SR: Methods in this category use destination addresses to indicate routing. Typically this involves rewriting the address at edge nodes so that the source and destination hosts use a native address independent of the network structure. This approach can use addresses at various levels of the protocol stack, including MAC and IP addresses. Work in this category includes [5]–[9], [22], [41] and [36].

SR may be coupled with an SDN controller, which has a view of the overall network topology and may have insight into its status as well. Such an SDN controller can compute routes across the network and enforce those routes by configuring either the source routes inserted into packets or to the forwarding tables, or both. In the literature, SR associated with SDN mostly follows the label-based approach, perhaps since MPLS allows a straightforward implementation of SR. In some cases, such as in the methods of [28] or [33], labels expose the source of the packet to many or all possible paths towards the destination. Imposing a route in the label reduces the forwarding table burden and can improve throughput.

Label-based SR has also been used to provide backup routes and perform traffic engineering. [44], presents an SDN-based Fast Rerouting Mechanism (SFRM) for rescheduling elephant flows in a load-aware manner. A flow-selecting module in the controller reschedules elephant flows selectively and reroutes a few elephant flows with SR to balance the load. In SlickFlow [38], packet headers contain a

[1]`https://tools.ietf.org/wg/spring/`

source route for a primary path and an alternate path. In these approaches, substantial path information is included in the packet header, which expands accordingly. Moreover, these techniques require changes in the core of the network to handle the new headers (although this can be minimized if an MPLS data path is specified). Another set of work uses SR to perform traffic engineering with the objective of reducing the power consumption of the data center. [23], [29], [37] and [30] are similar works using label-based SR (with MPLS) to aggregate traffic and shut down unused links within data centers, therefore saving energy. The SDN controller specifies links to be decommissioned and the label-path mappings in the switch forwarding tables.

Address-based SR carries routes in MAC or IP addresses, or a combination. Typically, this uses a destination address to identify a routing segment, with address translation or modification typically required at the termination of that segment. Some publications also specify interactions with other protocols, such as ARP, to provide routing addresses. For example, the Monsoon [5] data center architecture is designed to connect 100,000 or more servers using a single Layer 2 network. In order to overcome scaling limitations, the network is made hierarchical by encapsulating frames using three MAC addresses, which specify three particular switches along the route. The three addresses are inserted at the source server, based on a customized directory lookup; subsequent forwarding decisions are made by switches based on table lookups. The use of three MAC addresses introduces significant frame overhead and also operational complexity due to encapsulation and decapsulation and the directory service required to support it.

PortLand [6] presumes a Clos network and achieves a hierarchy without MAC encapsulation by assigning a hierarchical 48-bit Pseudo MAC (PMAC) address encoding the end host zone. End hosts maintain a separate Actual MAC (AMAC) address. ToR switches perform the address translation and header rewriting. PortLand employs a logically centralized fabric manager to maintain a soft state of the network configuration in order to construct the PMACs and respond to ARP replies. Torii-HLMAC [7] assigns to each node a Hierarchical Local MAC (HLMAC) address at every port. Torii-HLMAC improves PortLand by automatically assigning multiple addresses in a distributed form without duplicates, avoiding the use of a centralized fabric manager. GA3 [8] is a generalized labeling protocol for data center networks, originally designed for the Torii-HLMAC [7] routing protocol but later extended to support a wider range of topologies based on the same concepts.

DCnet [9] uses a hierarchical, zonal MAC address, known as a Routable MAC (RMAC) address, shared among all VMs in a server. DCnet also identifies each VM with a Unique ID. A logically centralized binding server keeps track of the mapping of these Unique IDs to the RMACs, distributing the mapping to all the ToR switches in the data center. When a VM migrates among servers, its Unique ID remains but a new zonal RMAC address is assigned and the update is distributed to all the ToR switches. [41] shows how the destination MAC address can be used as universal label in SDN and how the ARP cache of a host can be exploited as an ingress label table and therefore reduce the size of the forwarding table. This offloads packet labeling to the host, using the ARP protocol, rather than requiring a forwarding table entry to form the label at the ingress switch.

Shadow MACs [22] uses addresses as opaque forwarding labels, allowing an SDN controller to leverage large MAC forwarding tables to manage a plethora of fine-grained paths. In this shadow MAC model, the SDN controller can install MAC rewrite rules at the network edge to guide traffic onto intelligently selected paths for traffic balancing, to avoid failed links, or to route flows through middleboxes.

Finally, [36] proposes an OpenFlow (OF)-based Scalable Routing strategy (OSCAR) for DCNs using a hybrid addressing mechanism. Each module in the DCN constitutes a segment in the network. Inter-segment routing is performed using virtual MAC IDs assigned to the segments and intra-segment routing is done using IP addresses.

In general, SR requires mechanisms to express a path in a label or address. Arithmetic-based SR approaches, explored in several publications, make use of arithmetic operations, such as a Residue Number System (RNS). This approach was first applied to optical packet-switched networks to avoid header rewriting and label distribution protocols [43]. This idea was further integrated with SDN in core packet-switched networks by KeyFlow [35], which builds a fabric model to replace the table lookup in the forwarding engine by elementary operations relying on RNS. Another approach to RNS [39], applied to Service Function Chaining (SFC), includes two RNS labels, one for routing in the physical layer and one for routing between Service Functions in an overlay layer. KeySet [40] presents a new routing scheme without flow tables that enables constant-time switching at the forwarding switches. KeySet relies on a residual system to quickly calculate routing paths. A further elaboration of this concept, KeySFC [26], uses simple label-based SR in edge software switches to classify, encapsulate, forward, and decapsulate flows, along with core table-less switches that forward packets based on simple modulo operations over the labels. Finally, [25] proposes an algorithmic-based SR forwarding making use of a high-performance forwarding mechanism based on XOR operations. The major drawbacks of these solutions are the need for computation at the switches and possible scalability issues in large networks.

Another possible mechanism to embed the path information into the label or address is to represent the route as a sequence of ports, using Port Switching Source Routing (PSSR). Since the network topology of the data center network is typically structured and static by design, PSSR represents a routing path as a sequence of switch egress ports leading to a known destination. For example, Torii-HLMAC [7], using address-based SR, usually works as PSSR since addresses convey a sequence of traversed ports. The LESS method [11] embeds egress port identifiers directly in

TABLE 1. Related data center switching methods.

Switching Method	SR type / Path	Topology	Key contributions	Drawbacks
Monsoon [5] (2008)	Address-based/ Address stack	Any hierarchical network	Hierarchical scalable routing	Frame overhead + Computational complexity
PortLand [6] (2009)	Address-based/ Pseudo MACs	Specific Clos networks	Automatic address assignment and path repair based on SDN controller	Logically centralized control server
MAC addresses as routing labels [41] (2014)	Address-based/ Destination Address	Any network	Reduced table size with standard MAC addresses	Asymmetric address communication
Shadow MACs [22] (2014)	Address-based/ MAC as forwarding label	Any network	Addresses as opaque forwarding labels for fine-grained routing control	Requires different forwarding mechanisms for edge and core nodes
DCnet [9] (2017)	Address-based/ Routable MACs	Any hierarchical network	Transparent VM mobility	Logically centralized control server
OSCAR [36] (2017)	Address-based/ Hybrid addressing (MAC&IP)	Any hierarchical network	Hybrid addressing (L2+L3) for intra and inter-segment routing	Non-optimal shortest path routing
VirtPhy [25] (2017)	Address-based/ Arithmetic-based	Any network	Focus on SDN/NFV-based edge data centers	Computational complexity + Limited network size
Torii-HLMAC [7] (2012)	Address-based/ PSSR	Any hierarchical network	Table-free routing + on-the-fly path repair	Specific protocol for address distribution
GARDEN [10] (2012)	Label-based/ Arithmetic-based	Any hierarchical network	Multipath forwarding and good fault tolerance	Forwarding tables should carefully populated to avoid big tables
SlickFlow [38] (2013)	Label-based/ Arithmetic-based	Any hierarchical network	Augmented fault tolerance carrying alternative paths in packet headers	Frame overhead
SFRM [44] (2017)	Label-based/ Label stack	Specific Clos networks	Fast rerouting for elephant flows	Frame overhead
KeySet [40] (2017)	Label-based/ Arithmetic-based	Any network	Table-free routing (simplified hardware)	Computational complexity + Limited network size
SecondNet [31] (2010)	Label-based/ PSSR	Any hierarchical network	Creation of dynamic virtual data center adjusted to specific tenants' needs	Requires the configuration of MPLS
Sourcey [32] (2016)	Label-based/ PSSR	Any network	Table-free routing (simplified hardware)	Frame overhead + Label push/pop cost
LESS [11] (2017)	Label-based/ PSSR	Any network	Table-free routing (simplified hardware) + on-the-fly path repair	Frame overhead + Label push/pop cost
Zhao et al. [45] (2019)	Label-based/ PSSR	Any hierarchical network	Table-free routing (simplified hardware)	Frame overhead + Label push/pop cost
Gonzalez-Díaz et al. (our proposal)	Label-based/ PSSR/ SDA	Any hierarchical network	Table-free routing (simplified hardware) + No frame overhead	Limited to hierarchical networks

MAC addresses and routing instructions as a label-switched path. This approach eliminates the need for forwarding tables but requires extra overhead in the frames for the labels and effort at the switches in order to push and pop labels. Sourcey [32] proposes an oversimplification of the DCN in which switches have no CPU, no software, no forwarding tables, no state, and no configuration. Sourcey pushes all control plane functions, including routing, to the servers. At each hop, a Sourcey switch pops the top label of the path stack and uses the label value as the egress port identifier. Although this mechanism highly simplifies the switches, it requires overhead per packet and modification of each packet. SecondNet [31] is a DCN virtualization architecture based on PSSR, implemented using MPLS. In order to handle the case of a server connecting to multiple neighbors through the same port, SecondNet introduces the concept of virtual port. A physical port can map to multiple virtual ports depending on the number of neighboring servers. A server maintains a virtual-port table in which every row represents a neighboring server. Finally, [45] proposes a port-based source routing addressing scheme that renders table lookup unnecessary and can reduce the complexity of the switches, taking advantage of fat-tree topology.

To summarize this literature, Table 1 provides a tabular view of a representative set of data center switching methods, characterizing each according to its SR type, the presumed topology, some key contributions, and some identified drawbacks.

IV. SOFTWARE-DEFINED ADDRESSING

The key rules of IEEE 802 MAC addresses are specified in IEEE Std 802 [14], which covers the IEEE 802 *Overview and Architecture*. This standard specifies that the least significant MAC address bit is the Individual/Group (I/G) bit, used to identify the destination address either as an individual or as a group address, and the second least is the Universally or Locally Administered (U/L) bit, indicating whether the address has been assigned for global or only local uniqueness. Regarding locally administered addresses, virtually no information was provided until 2017, leaving implementation and use completely in the hands of the local administrator, with no interoperability between protocols mandated.

Since 2017, an amendment to IEEE Std 802, entitled IEEE Std 802c [15], has added specifications regarding local MAC addresses, including a plan, known as the Structured Local Address Plan (SLAP), for use of the local address space. The SLAP divides the local space into four quadrants, each of which is specified for a different use. The quadrants (identified by the third and fourth least significant bits of the initial octet in the local MAC address) are identified for use by:

1) **Extended Local Identifier (ELI) addresses**, which are 24-bit extensions of a 24-bit Company ID (CID) assigned to a company upon application to IEEE. Such addresses are similar in structure to the historical 48-bit Extended Unique Identifier (EUI) addresses based on IEEE-assigned 24-bit Organizationally Unique Identifier (OUI) identifiers. The difference is that OUI assignments are made to hardware vendors with the intention of the EUI-48 being permanently assigned to hardware. The ELI-48 is expected to be locally, but not necessarily globally, unique and can be assigned dynamically;
2) **Administratively Assigned Identifier (AAI) addresses**, which assigned in arbitrary fashion by an administrator, with simply a requirement to avoid duplication; and
3) **Standard Assigned Identifier (SAI) addresses**, whose specified use is, per IEEE Std 802c, the responsibility of the future IEEE 802.1CQ [16] standard. Multiple protocols for assigning SAI may be specified within various IEEE 802 standards. Per IEEE Std 802c,"In some cases, an SAI assignment protocol may assign the SAI to convey specific information. Such information may be interpreted by receivers and bridges that recognize the specific SAI assignment protocol, as identified by the subspace of the SAI."

While a closed network may operate successfully without observing the SLAP, the standardization progress nevertheless highlights the trend toward assigning MAC addresses with semantic meaning and ensuring that switches understand the semantics and process frames accordingly. We use the term "software-defined addressing" to describe techniques in which Layer 2 addresses are semantically structured, rather than flat, and dynamically assigned, rather than hardware bound, with the addresses serving to steer frames through the infrastructure. In this paper, we intend to explore whether software-defined addressing, with semantic cues to routing and QoS management, can increase network performance. At the same time, we surmise that it can significantly reduce the complexity and operational expenses of the network switching infrastructure while improving latency at the switch by minimizing processing requirements.

V. FLOW-ZONE SWITCHING: STRUCTURE

Flow-zone switching (FZS) [17] is a loop-free routing method that embeds not only routing instructions but also flow identity directly in addresses within the frame. Given a suitable DCN topology and a compatible assignment of addresses, forwarding tables are not required. In this paper, we consider FZS with Layer 2 addressing.

FZS is compatible with many multi-tier topologies, including generalized Clos data center architectures. Such topologies provide multiple paths between hosts to support full bisection bandwidth. This paper analyses the method in a pod-spine Clos architecture [2].

A. PRINCIPLES

The following key properties are characteristic of FZS:

- **Zonal addressing**: Each FZS end node is assigned a set of flow-zone addresses, coded within which are zonal identification fields that uniquely identify the zone; that is, the location of the end node within the data center topology, including information that encodes forwarding instructions to reach that zone.
- **Flow addressing**: The FZS addresses assigned to the end node are distinguished with flow identification fields embedded in the flow-zone address. A flow-zone address may include multiple flow identifiers. Flow identifiers may be used within the switching networks, for example as flow type indicators for purposes such as QoS. They may also be used at the end nodes, for example, as indicators of the virtual machine destination within the node and as indicators of suppressed headers.
- **Frame addressing**: Each frame carries an Source Address (SA) that is assigned by source end node from its allocated set of flow-zone addresses. Each also carries a flow-zone Destination Address (DA). Flow identification fields in both the SA and DA may be considered by switches.
- **Switches**: Flow-zone switching is based on flow-zone addresses. Switches are aware of their own zonal identification fields, which indicate their position in the topology. Forwarding tables, if provided, are simple port mapping tables serving only to translate a specific field of the flow-zone address into an output port. Such tables are therefore extremely small: typically, one entry per local port, rather than scaling with the number of hosts. In some cases, the addressing is configured so that the specific field of the flow-zone address is a literal identifier of the output port, in which case no forwarding tables are used.
- **Routing**: FZS routing is based on forwarding decisions made by switches. Data frames, generated at end nodes, begin in an ascent stage during which they are forwarded

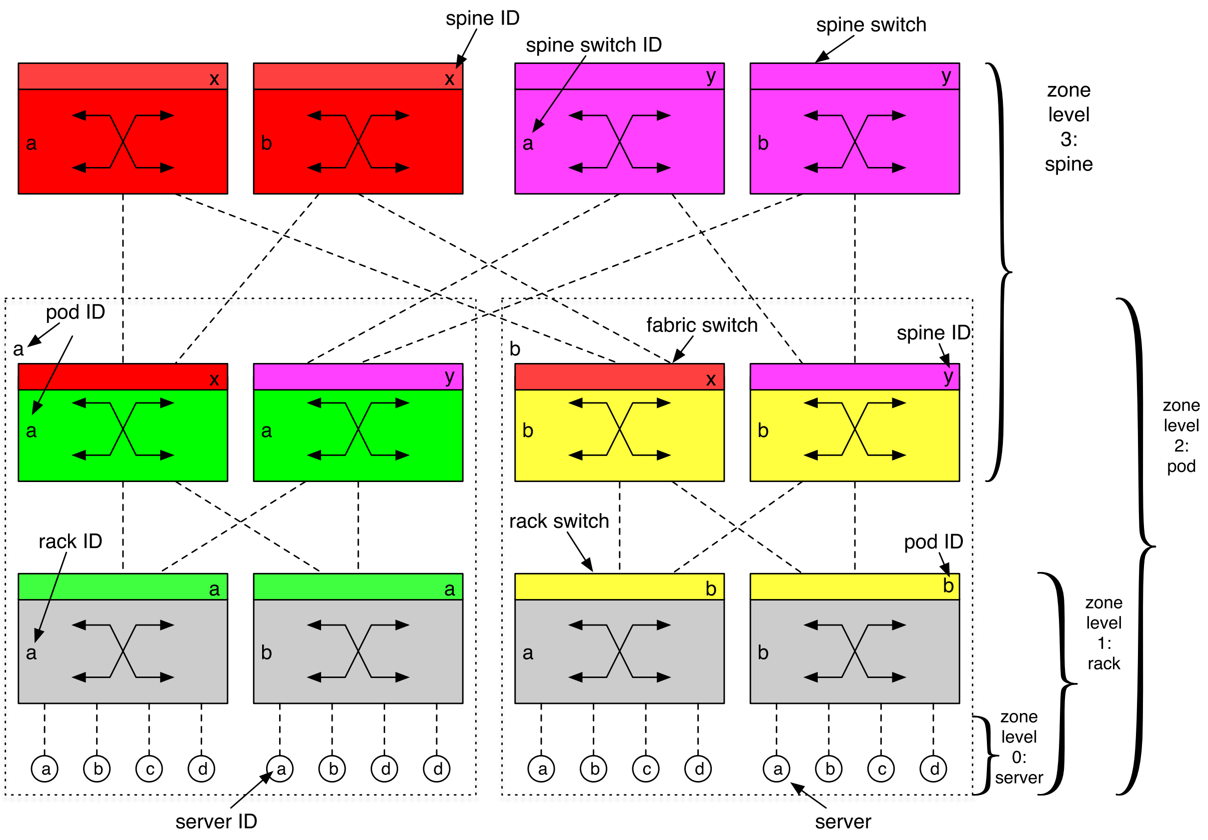


FIGURE 1. Zone levels.

upward in the network, typically until they have risen high enough to begin descent down through the topology to the destination zone. During the ascent stage, multiple equal-cost routes are possible. The choice among these routes is informed by the flow identifiers within the flow-zone addresses. Frames belonging to a common flow, indicated by suitable flow identifiers, are routed on a common path. During the descent phase, typically only a single lowest-cost path is available, and the routing follows that path, typically based solely on the zone identification fields of the destination flow-zone address.

B. TOPOLOGY, NODES, AND ZONE LEVELS

A flow-zone address includes fields identifying a zone within the topology. This paper considers the pod-spine Clos topology (see Fig. 1).

The zone is identified with reference to a set of zone levels, including, in the pod-spine Clos topology, five zone levels (see Fig. 1):

- **Zone Level 0**: The lowest level of the FZS topology, Level 0, represents the servers, i.e., the end nodes of the data center traffic, represented by the circles in Fig. 1. Each server is connected to a top-of-rack (ToR) switch and is identified by a *Server ID* (“a”, “b, “c”, and “d” in the figure) unique among all the servers connected to the same rack. In order to fully identify a server in the topology, the *Server ID* is used in conjunction with a unique identifier of the rack to which it is connected.
- **Zone Level 1**: Level 1 represents the racks, each including the servers and the ToR switch to which they are connected, as colored in gray in Fig. 1. Each rack exists within one and only one pod (see Level 2) and is identified by a *Rack ID* (“a” and “b” in the gray areas of the figure) unique among all the racks within that pod.
- **Zone Level 2**: Level 2 represents the pods, indicated in the figure by dotted lines. A pod includes the racks within the pod as well as a set of fabric switches, shown as the green and yellow switches in Fig. 1. Each fabric switch exists within one and only one pod. Each pod is identified by a *Pod ID* (“a” in the green fabric switches and “b” in the yellow fabric switches of the figure) unique among all the pods within the network. Each ToR switch in the pod is connected to each fabric switch in that pod.
- **Zone Level 3**: Level 3 is the spine level. Each spine switch (the red and magenta switches of Fig. 1) is connected to one fabric switch in each pod. A spine switch, together with all the fabric switches to which is connected, comprise a spine. Each spine switch and each fabric switch exists within one and only one spine and is identified by a *Spine ID* (“x” and “y” in the red and

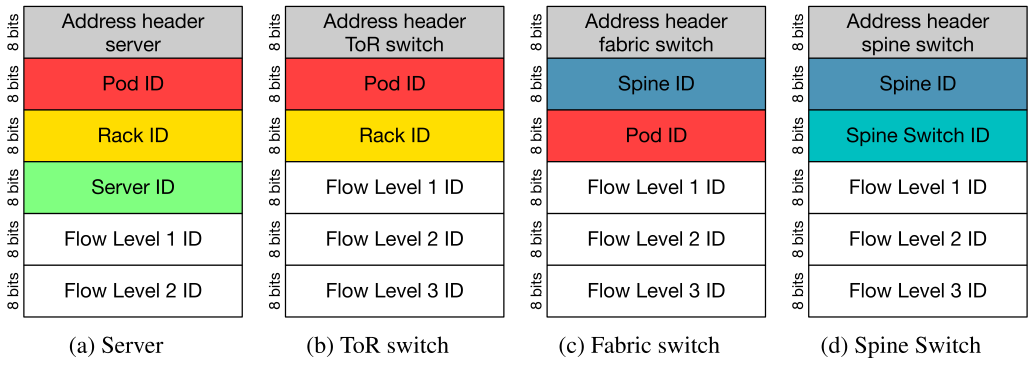


FIGURE 2. Flow-zone address formats.

magenta areas of the figure) unique among all the spines in the network.

- **Zone Level 4**: Level 4 (not identified in the figure) is the spine switch level. A spine switch is identified by a *Spine Switch ID* (“a” and “b” in the red and magenta spine switches of the figure) to uniquely identify it within its spine.

C. DEVICE IDENTIFIERS AND ZONES

Each node in the FZS infrastructure is identified with a zonal tuple that uniquely identifies both the zone and its location in the network. Fig. 1 depicts an example of zonal tuple assignment in a small topology of 12 switches and 16 servers. The zonal tuple of each switch includes two independent identifiers. The zonal tuple of each server includes three independent identifiers. Each node is aware of its own node type and of its own zonal tuple. The node types are:

- **Server**: The server is located in a zone that is specified by its identifier at Zone Level 2 (*Pod ID*), Zone Level 1 (*Rack ID*), and Zone Level 0 (*Server ID*).
- **ToR Switch**: The ToR switch is located in a zone that is specified by its identifier at Zone Level 2 (*Pod ID*) and Zone Level 1 (*Rack ID*).
- **Fabric Switch**: The fabric switch is located in a zone that is specified by its identifier at Zone Level 2 (*Pod ID*) and Zone Level 3 (*Spine ID*).
- **Spine Switch**: The spine switch is located in a zone that is specified by its identifier at Zone Level 3 (*Spine ID*) and at Zone Level 4 by its *Spine Switch ID*.

D. FLOW-ZONE ADDRESS FORMATS AND ZONAL INDICATOR FIELDS

Each node is assigned a set of addresses for use as source and destination addresses. Each such address includes fields to identify the complete zonal tuple of the node to which that address is assigned. In the examples of this paper, flow-zone addresses are IEEE 802 MAC (Layer 2) addresses, typically in the local address space. For simplicity of explanation, and in accordance with the typical DCN scales, the 48-bit IEEE MAC address is divided into six octets. The first of these contains an address header. Per IEEE Std 802, least significant bits of the address header are, respectively, the M bit (set to 1 for a multicast address), the X bit (set to 1 for a local address), and two bits indicating the SLAP quadrant. The remaining bits of the address header are used to indicate the node type of the device to which the address is assigned. The flow-zone addresses are illustrated in Fig. 2 for the node types discussed herein.

- **Server Address Format**: In the server flow-zone address, the address header identifies the address as belonging to a server. Additional octet-sized fields identify the server’s zone by its *Pod ID*, *Rack ID*, and *Server ID* (see Fig. 2a).
- **ToR Switch Address Format**: In the ToR switch flow-zone address, the address header identifies the address as belonging to a ToR switch. Additional octet-sized fields identify the switch’s zone by its (*Pod ID*) and *Rack ID* (see Fig. 2b). (Alternately, the ToR Switch and Server addresses could use identical address header formats and be distinguished by, for example, a particular value in the *Server ID* field.)
- **Fabric Switch Address Format**: In the fabric switch flow-zone address, the address header identifies the address belonging to a fabric switch. Additional octet-sized fields identify the switch’s zone by its *Pod ID* and *Spine ID* (see Fig. 2c).
- **Spine Switch Address Format**: In the spine switch flow-zone address, the address header identifies the address belonging to a spine switch. Additional octet-sized fields identify the switch’s zone by its *Spine ID* and *Spine Switch ID* (see Fig. 2d).

Bits in the flow-zone address not devoted to zonal identifier fields are available for use in flow identification. In Fig. 2, the flow identification bits are indicated as divided into one-octet fields, but the actual division can be made flexibly. As a practical example, in the server flow-zone address of Fig. 2a, the Flow Level 1 ID field could be a one-octet field specifying the frame’s virtual machine (VM, which here also includes containers) source or destination, and the Flow Level 2 ID field could represent a specific flow of frames originating from or destined for that VM. The flow identification fields are fully flexible and may be used to classify frames according to categories that may not normally be considered "flow" categories; furthermore, those fields could carry any descriptive information, including, for example, telemetry. The flow fields are directly exposed at Layer 2, without requiring deeper inspection.

The order of zonal identifier fields in the flow-zone address is not critical, but it may be convenient for all flows associated with a device to share a common preamble and to be discriminated by the less significant bits of the address; in this case, the device is assigned a contiguous block of addresses.

The dimensions of the zonal identifier fields can be adjusted to match the scale of the network. The dimensions of Fig. 2 would support, for example: 256 spines, 256 spine switches per spine (each with 256 ports), 256 pods, 256 racks per pod, 256 fabric switches and 256 ToR switches per pod (each with 512 ports), and 256 servers per rack, for a limit of 2^{24} ($\approx$17M) total servers. The flow partitioning as described supports 256 VMs per server, with 8 bits remaining for flow differentiation within the VM. This would support a network vastly larger than those currently deployed or anticipated. In a smaller network, the zonal fields can be compacted, freeing up bits in the address for flow fields.

VI. FLOW-ZONE SWITCHING: FORWARDING

A. FLOW-ZONE FORWARDING TO SERVER

Flow-zone routing can be summarized by rules governing the forwarding by each node type.

In general, the device first examines the address header of the destination address of the incoming frame to determine the flow-zone address format, as illustrated in Fig. 2. Here we describe the forwarding process only for the case in which that address header indicates a server (Fig. 2a). Similar considerations provide for appropriate forwarding when the destination is a switch. If the destination header indicates that the destination address is *not* formatted as a flow-zone address, the switch may process it with an alternative forwarding method; e.g., using legacy Layer 2 forwarding.

- **Server Forwarding**: A server compares the *Pod ID*, *Rack ID*, and *Server ID* of the destination address of an incoming frame to its own internal identities. In case all three parameters match, the frame is passed to a local process, such as a local VM as identified by the Flow Level 1 ID parameter. If the three parameters do not match, the server forwards the frame upward to its ToR switch.
- **ToR Switch Forwarding**: A ToR switch compares the *Pod ID* and *Rack ID* of the destination address of an incoming frame to its own internal identities. In case the parameters match, a port mapping is used to determine the port to the server identified by the frame's *Server ID* value; the frame is forwarded downward to that port. If the two parameters do not match, the switch forwards the frame upward to a port attached to a fabric switch. The choice of ascending port may be informed by the frame's flow identification values.
- **Fabric Switch Forwarding**: A fabric switch compares the *Pod ID* of the destination address of an incoming frame to its own internal *Pod ID* identity. In case the parameters match, a port mapping is used to identify the port to the rack identified by the frame's *Rack ID* value; the frame is forwarded downward to that port. If the *Pod ID* values do not match, the switch forwards the frame upward to a port attached to a fabric switch. The choice of ascending port may be informed by the frame's flow identification values.
- **Spine Switch Forwarding**: A spine switch uses a port mapping to identify the port to the pod identified by the frame's *Pod ID* value; the frame is forwarded downward to that port.

Notice that, in all cases, the incoming port need not be identified in order to complete the forwarding decision, and no determination need be made as to whether the frame arrived an ascending or descending frame.

B. STATELESS ZONAL FORWARDING TO SERVER

As seen from the description of flow-zone forwarding, no address learning is used, and the only forwarding tables are the port mapping tables indicating the port of the identified server, rack, or pod. These are at most small tables, with one entry per active port. This requires a very small amount of memory and a simple one-to-one lookup.

The port mapping can be made entirely stateless as well. To achieve statelessness, the zonal identifiers can be selected so that they literally identify, in numbers directly meaningful to the switch, the ports to the identified zones. Achieving this result requires specific network cabling. Fig. 3 illustrates this cabling method and the associated zonal identification. The small white rectangles indicate ports, and the letters therein indicate the internal identifier with which the switch identifies each port. Fig. 3 illustrates a single pod (*Pod ID*=*J*) and its connections.

In the figure, the rack switches and fabric switches are shown with an upper bank of ports, with an identifier beginning with "1", and a lower bank of ports, with an identifier beginning with "0". This requires an explanation. As noted earlier, with 8-bit zonal identification fields, the network can scale to 256 racks per pod and 256 spines. In this case, the fabric and racks switches need to support 512 ports: 256 upwards and 256 downwards. The port numbering in the figure is designed to avoid limiting the network scale. Let us assume that 512 ports are numbered 0-511. The lower half of these (0-255, with the most significant bit of the identifier equal to 0) are cabled downward; the upper half (256-511, with the most significant bit of the identifier equal to 1) are cabled upward. In this case, the nine-bit port identifier can be abbreviated by its eight least significant bits (e.g., 0A and 1A are both abbreviated as "A"). However, from the switching context, the switch can always determine whether the identified port

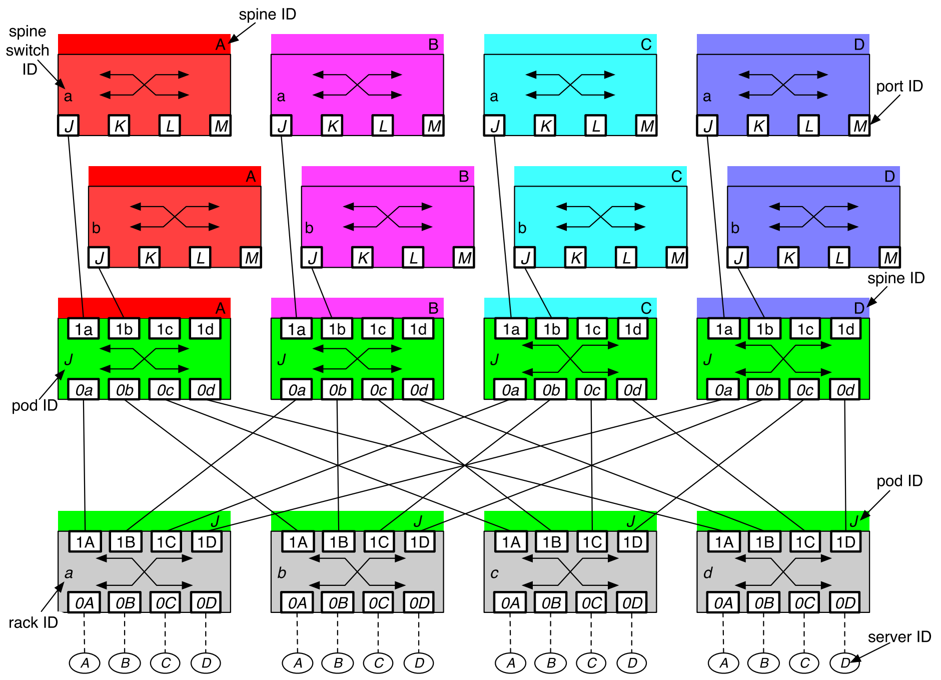


FIGURE 3. Cabling and identifiers for Stateless Flow-Zone Forwarding.

is upward or downward, as confirmed by review of the rules in subsection VI-A for forwarding of a server-directed frame. Therefore, using this technique, the network can scale to full dimension. If the rack switches and fabric switches are limited to 256 ports, the abbreviation is not necessary, but little effort is saved by doing away with it.

Fig. 3 is helpful in understanding these key points of the stateless configuration:

- **Server**: The local zonal identify of each server (*Server ID*, equal to *A*, *B*, *C*, or *D* in the ovals of the figure) is identical to the (abbreviated) internal port identifier, within the rack switch, of the port at which the server is connected. This suffices to eliminate the need, described in subsection VI-A, for a port mapping table to identify the port associated with a *Server ID* value.
- **Rack**: The local zonal identify of each rack (*Rack ID*, equal to *a*, *b*, *c*, or *d* in the figure) is identical to the (abbreviated) internal port identifier, within the fabric switch, of the port at which the rack's ToR switch is connected. The cabling condition is that every link to a rack from a fabric switch must be connected, at the fabric switch, by an identically named port. For example, in Fig. 3, every link from a fabric switch to the ToR switch in *Rack ID*=*d* is connected to that fabric switch's port *0d*. This suffices to eliminate the need, described in subsection VI-A, for a port mapping table to identify the port associated with a *Rack ID* value.
- **Pod**: The zonal identify of each pod *Pod ID* (*J* in the figure) is identical to the internal port identifier, within the spine switch, of the port at which the pod's fabric switch is connected. The cabling condition is that every link to a fabric switch from a spine switch must be connected, at the spine switch, by an identically named port. For example, in Fig. 3, every link from a spine switch to any fabric switch in *Pod ID*=*J* is connected to that fabric switch's port *J*. This suffices to eliminate the need, described in subsection VI-A, for a port mapping table to identify the port associated with a *Pod ID* value.

These conditions suffice to eliminate all three of the port mapping tables and the associated zonal lookup requirements during frame descent toward a destination server. Consider, for example, a frame sent from outside pod *J* to the rightmost server (server *D* of rack *d*) of Fig. 3. The frame will ascend to a spine switch, which will check the destination *Pod ID*, see the value *J*, and then, without a table lookup, forward the frame out port *J*, forwarding it thereby to pod *J*. The frame will arrive at a fabric switch, which will determine that the destination *Pod ID* identifies the fabric switch's own pod identifier (pod *J*) and therefore that the frame needs to descend. Then, checking the destination's *Rack ID*, the fabric switch finds the identifier *d*. Without a table lookup, the fabric switch forwards the frame downward to its port *0d*, which delivers it to rack switch *d*. Finally, rack switch *d* will determine that the destination *Pod ID* and *Rack ID* jointly identify the rack switch's own rack identity and therefore that the frame needs to descend. Checking the destination's *Server ID*, the rack switch finds the identifier *D*. Without a table lookup, the rack switch forwards the frame downward to its port *0D*, which delivers it to server *D*. This forwarding system works regardless of the spine switch at which the descent began.

Note that the port mapping tables (or alternatively the stateless flow-zone zone identification which avoids those tables) are relevant only to routing the frame toward the server during the descent phase. During the ascent phase, the forwarding decisions are not based on the destination zone, since any fabric switch and any zone switch will be suitable.

C. STATELESS ZONAL FORWARDING TO SWITCHES

The prior subsection considered only the forwarding of frames to servers, in which case the choice of spine and spine switch may be secondary or arbitrary and in any case is relevant only during descent. As noted above, this paper does not fully detail the forwarding processing of frames directed toward switches, but here we do note that fabric switches (Fig. 2c) and spine switches (Fig. 2d) are located on a particular spine, so identifying the *Spine ID* is a critical step in the routing process. Likewise, the *Spine Switch ID* is essential to routing a frame to a spine switch. The process of forwarding such frames can make use of small port mapping tables, as described above for the routing of server-directed frames, or those lookups can be eliminated with stateless forwarding.

The cabling and zonal identification structure of Fig. 3 is designed to eliminate port mapping tables for switch-directed frames as well as server-directed ones. The key additional points illustrated in the figure are:

- The zonal identity of each spine switch *Spine Switch ID* is identical to the (abbreviated) internal port identifier, within the fabric switch, of the port at which that spine switch is connected. The cabling condition is that every link to a spine switch from a fabric switch must be connected, at the fabric switch, by an identically named port. For example, in Fig. 3, every link from a fabric switch to a spine switch with *Spine Switch ID*=a is connected to that fabric switch's port 1a. Note that the figure does not fully illustrate this point because only one pod is shown.
- The zonal identity of each spine *Spine ID* is identical to the (abbreviated) internal port identifier, within the ToR switch, of the port at which that spine switch is connected, via a fabric switch in that spine. The cabling condition is that every link to a fabric switch in a particular spine from a ToR switch must be connected, at the ToR switch, by an identically named port. For example, in Fig. 3, every link from a ToR switch to a fabric switch with *Spine ID*=A is connected to that fabric switch's port 1A.

Consider, for example, a frame addressed to the rightmost spine switch (spine *D*, spine switch *b*) of Fig. 3. If the rack switch determines, based on the address header, that the frame is of the type illustrated in Fig. 2c or Fig. 2d, then it reads the *Spine ID* value (D) from the destination address and accordingly forwards the frame upward to port 1D, which passes it to a fabric switch in spine D. If that fabric switch determines, based on the address header, that the frame is of the type illustrated in Fig. 2d and determines that the switch is on the same spine as the destination, then it reads the *Spine Switch ID* value (*b*) from the destination address and accordingly forwards the frame upward to port *1b*, which passes it to the destination switch.

D. FLOW FORWARDING

While stateless flow-zone zone identification for server-directed frames is relevant only in the descent phase, we can also consider flow-zone flow identification, which is relevant to the ascent phase. Namely, the rack switch, forwarding up to a fabric switch, needs to select a spine; likewise, the fabric switch, forwarding up, needs to select a spine switch within that spine. Those selections cannot be made completely arbitrarily; for example, the network must be constrained to maintain the order of frames within a flow, which implies that each frame in the flow should follow the same route. Equal-cost multi-path routing (ECMP) typically approaches the problems by computing a hash over some components of the frame sufficient to identify the flow and then forwarding based on the hash. This is a nonideal approach because (1) the flows are imprecisely identified by the network; (2) the flows are all treated equally and intermingled, without respect to their differentiated QoS requirements; (3) the hash calculations are an additional burden to the switches.

The FZS approach to this problem is to specify the flow in the Flow Level ID fields of the frame and ensure that the switches are informed of the preferred *Spine ID* and *Spine Switch ID* of that flow. The switch can be informed in various ways. For example, port mapping tables could be used to translate the Flow Level 2 ID and Flow Level 1 ID fields of a server-directed frame into specific ports leading to the designated spine and spine switch; this could also be made stateless (see the following subsection). With additional complexity, switches could be enabled to make independent decisions; for examples, choosing an alternative forwarding port based on knowledge of congestion.

E. STATELESS FLOW FORWARDING

The flow-zone zone identification of subsection VI-C can be applied to stateless ascent forwarding of server-directed frames based on flow identifiers. Consider, for example, a frame sent from the leftmost server (server *A* of rack *a* of Fig. 3). Once the rack switch determines, based on the destination address zonal fields, that the frame needs to be forwarded upwards, it needs to choose a port (*1A*, *1B*, *1C*, or *1D*). Using stateless flow-zone flow forwarding, that decision is made (or at least suggested) by the frame itself, which contains the literal port identifier. This could be contained in, for example, the Flow Level 2 ID field. So, for example, Flow Level 2 ID field value *D* would direct the rack switch to forward the frame upward out port *1D*, which would transfer it to a fabric switch in Spine *D*. From there, the fabric switch, reading (for example) Flow Level 1 ID field, might find the value *b* and therefore forward the frame out its port *1b*, thereby transferring the frame to Spine Switch *b*.

F. STATELESS FLOW-ZONE ADDRESS ASSIGNMENT

In order to further illustrate the network configuration required for stateless flow-zone forwarding, we provide here an algorithm to automatically assign, given the cabling arrangement of Fig. 3, all of the zone identifiers to match the port identifiers of the output ports leading to those zones. The cabling requirements are described in more detail above in subsections VI-B and VI-C.

All messages exchanged in conducting this algorithm are transmitted as link-local frames, meaning that they are not forwarded by a receiving switch. For example, IEEE Std 802.1Q specifies that frames addressed to 01:80:C2:00:00:0E are not forwarded by an IEEE 802.1Q bridge. These messages will be read only by the immediate recipient.

1) Spine switches are made aware that they are spine switches and should begin the assignment process.
2) Each spine switch sends a spine-to-fabric message from each of its ports indicating that the recipient is a fabric switch and including a parameter equal to the spine switch's internal identifier of that output port.
3) Each fabric switch receives those spine-to-fabric messages and confirms that they all contain the same value; otherwise, the physical connectivity is incorrect and a diagnostic error results. If the incoming messages are consistent, the fabric switch identifies those ports as upward spine switch ports, identifies itself as a fabric switch, and identifies its own *Pod ID* value as the common parameter received in the spine-to-fabric messages.
4) Each fabric switch sends, to each of its other live ports, a fabric-to-rack message with a Parameter 1 equal to its own *Pod ID* value and a Parameter 2 equal to the fabric switch's (abbreviated) internal identifier of the output port.
5) Each rack switch receives those fabric-to-rack messages and confirms that they all contain the same parameter values; otherwise, the physical connectivity is incorrect and a diagnostic error results. If the incoming messages are consistent, the rack switch identifies those ports as upward fabric switch ports, identifies itself as a rack switch, identifies its own *Pod ID* value as the common Parameter 1 received in the fabric-to-rack messages, and identifies its own *Rack ID* value as the common Parameter 2 received in the fabric-to-rack messages.
6) Each rack switch replies to each fabric switch with a rack-to-fabric message including a parameter equal to the rack switch's (abbreviated) internal identifier of that output port. Each fabric switch receives those rack-to-fabric messages and confirms that they all contain the same value; otherwise, the physical connectivity is incorrect and a diagnostic error results. If the incoming messages are consistent, the fabric switch identifies those ports as downward rack switch ports, confirms its identify as a fabric switch, and identifies its own *Spine ID* value as the common parameter received in the rack-to-fabric messages.
7) Each rack switch sends, to each of its other live ports, a rack-to-server message with Parameter 1 equal to its own *Pod ID* value, Parameter 2 equal to its own *Rack ID* value, and a Parameter 3 equal to the rack switch's (abbreviated) internal identifier of the output port.
8) Each server identifies those ports as rack switch ports, identifies itself as a server, identifies its own *Pod ID* value as the common Parameter 1 received in the rack-to-server messages, identifies its own *Rack ID* value as the common Parameter 2 received in the rack-to-server messages, and identifies its own *Server ID* value as the common Parameter 3 received in the rack-to-server messages.
9) Each fabric switch replies to each spine switch with a fabric-to-spine message including a Parameter 1 equal to its equal to its own *Spine ID* value and a Parameter 2 equal to the fabric switch's (abbreviated) internal identifier of the output port. Each spine switch receives those fabric-to-spine messages and confirms that they all contain the same value; otherwise, the physical connectivity is incorrect and a diagnostic error results. If the incoming messages are consistent, the spine switch identifies those ports as downward fabric switch ports, confirms its identify as a spine switch, identifies its own *Spine ID* value as the common Parameter 1 received in the fabric-to-spine messages, and identifies its own *Spine Switch ID* value as the common Parameter 2 received in the fabric-to-spine messages.

Each switch and server, once it has identified its zonal identifiers, configures a set of Layer 2 addresses for itself, based on the formats shown in Fig. 2.

VII. FLOW-ZONE SWITCHING: BENEFITS

To summarize key points of this description, flow-zone switching promises numerous benefits, including:

1) **Scalability**: As noted earlier, the version of flow-zone network described, with three one-octet zone identifier fields and two one-octet flow identifier fields in an address, can scale to 2^{24} servers with 256 VMs per server and 256 flows per server. In a smaller network, the zonal fields can be scaled to smaller size, and additional bits are thereby freed up for new purposes or finer flow granularity. In the near term, 2^{24} servers is far larger than necessary for practical data centers. For example, the hyperscale network of in [2] appears to use 4 spines (compared to a limit of 256), up to 48 spine switches per spine (compared to 256), and 48 racks per pod (compared to 256), with a topology claimed to be ‘‘capable of accommodating hundreds of thousands’’ of servers.
2) **Switch simplicity**: Flow-Zone switches, particularly with stateless configuration, maintain no forwarding tables and forward to ports specified in the frame,

potentially with discretion to use alternate forwarding when appropriate. This implies a drastic reduction in on-board memory. The parallel reduction the computational requirements (through the elimination of lookups, hash calculations, etc.) may result in measurable improvement in latency and electrical power consumption. Note that Layer 3 forwarding requires at least a frame update, replacing the IP TTL as well as the MAC SA, DA, and FCS fields; identifying the DA typically requires an ARP cache lookup, and the FCS requires recalculation. By comparison, Layer 2 forwarding requires none of these, as frames are forwarded unaltered.

3) **Overhead**: Because end-to-end routing is achieved directly based on Layer 2, extra frame overhead due to overlays is avoided. Also, flow identifiers can be used to as indices to headers that are common to all frames in the flow and may be consequently suppressed during transmission, to be rebuilt at the destination. This can achieve overhead reduction.
4) **Overhead**: Address distribution: Flow identifiers are exposed to all switches, allowing diverse flow management methods, as well as to the receiving end station, for post-reception processing such as header decompression indexed by the fine-grained flow. Each frame contains the path towards its source, as well as to the destination, with separate source-based and destination-based flow fields, assigned without the intervention of any other protocol or mechanism. This means, for example, that the destination can learn directly learn the Layer 2 flow-zone address associated with the IP address of a received frame and can respond to that frame without the need to consult an external ARP server.
5) **Network QoS**: Flows can be directly identified in the frame. The network can consequently route different flows differently, with fine granularity, based on instructions imposed at the source.

The issue of network QoS is a complex one that deserves further study and is, in fact, a primary subject of the evaluation study reported below. The example we have studied considers TCP flows. This is a simple but interesting case. TCP flows generally take two different forms: large data packets in one direction, and small, latency-sensitive acknowledgements (ACKs) in the other. In this paper, we refer to the data flows as ‘‘elephant flows’’, regardless of the number of frames in the flow, and the ACK flows as ‘‘mice flows’’. Likewise, we refer to data frames, typically 1500 bytes long, as ‘‘elephant frames’’, and ACK frames, typically 64 bytes long, as ‘‘mice frames’’. Switches that attempt to classify frames on the basis of typical frame headers (MAC addresses, IP addresses, Ethertype, Layer 4 port, etc.) will not see a difference between the elephant data frames and the mice ACKs. However, the source is well equipped to identify the ACKs and can assign them a flow identifier to distinguish them from the elephants. If the flow-zone network is configured to sense the flow identifiers and consequently route the mice onto a path that is reserved for mice only, then we can expect less-congested and lower-latency receipt of ACKs, which may result in a more efficient TCP process. Such a reserved route is easily configured, by steering the mice onto a particular spine or to particular spine switches, as described in VI-D. In such a simple example, one might ask whether the loss of bandwidth available to the elephants (due to their confinement to fewer spine switches) might be more than offset by the gain due to the timelier ACK delivery. Such a study is reviewed below. Note that this example is but one simple case of how FZS might be applied in the DCN for flow segregation.

VIII. FLOW-ZONE SWITCHING: IMPLEMENTATION

The SDN paradigm decouples the control and data planes, centralizing the network logic in a controller. SDN boosts network flexibility and programmability, as compared to legacy distributed-control networks. Control in SDN is installed in the data planes via diverse protocols and languages. We review the two most popular, which we considered for our FZS study: OpenFlow and P4.

A. OPENFLOW CONSIDERATIONS

The understanding and deployment of SDN has been significantly accelerated by the Open Networking Foundation (ONF) [19]. The original focus of the ONF was the specification of the OpenFlow Switch Speci?cation [18].

An OpenFlow Logical Switch includes ?ow tables that perform packet lookups and forwarding and provides an OpenFlow channel to an external controller to control and configure the flow tables. Lookups and actions are programmable and can be performed consecutively. While we considered implementing FZS in OpenFlow, we ultimately concluded that it was not sufficiently flexible.

An OpenFlow flow table can be programmed to match any bits, as specified by an arbitrary bit mask, of a frame’s MAC SA or DA. Based on this capability, we provide the following example of steps that could be implemented in OpenFlow to embody the ToR Switch Forwarding of a server-directed frame, as described in subsection VI-A:

- In the first flow table, the OpenFlow ToR switch compares the address header of the incoming frame’s DA to the address header indicating a server (Fig. 2a). In case of a match, the frame is passed to a second flow table.
- In the second flow table, the switch compares the *Pod ID* and *Rack ID* of the DA to the switch’s own internal identities. In case the parameters match, the frame needs to be forwarded downward to a server port; to determine the port, the frame is passed to a third flow table.
- The third flow table includes an entry for each *Server ID* and the associated identifier the port to that server. The switch checks for a match of the *Server ID* and the frame is forwarded downward to the identified server port.

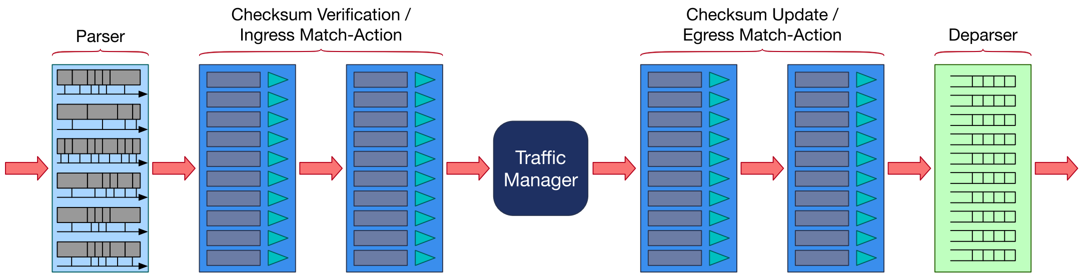

Parser
Checksum Verification / Ingress Match-Action
Traffic Manager
Checksum Update / Egress Match-Action
Deparser

FIGURE 4. **P4 abstract forwarding model.**

While this approach is generally suitable for implementing FZS, it cannot implement the stateless FZS of subsection VI-B because the last table needs to be stored and populated. In stateless FZS, the *Server ID* literally provides the port identifier. To our understanding, OpenFlow does not provide a means to forward to a port identified by a variable (such as a parameter read from the frame) but only to a value identified by a table match. As a result, we did not pursue an OpenFlow implementation of FZS.

B. p4 CONSIDERATIONS

OpenFlow evolved rapidly beginning in around 2008 but has not been updated since April 2015. Meanwhile, the ONF [19] hosts the ongoing development of P4.

P4 [20] (a name derived from the term ''programming protocol-independent packet processors'') provides a programmable language to specify the forwarding behavior of a switch, whether the switch is based on Application-Specific Integrated Circuit (ASIC), Network Processing Unit (NPU), Field Programmable Gate Array (FPGA) or reconfigurable soft switch technology. P4 is based on an abstract forwarding model, illustrated in Fig. 4, comprising a packet parser, a set of tables with match-action flows, and a deparser. The imperative P4 language describes the behavior of each of the abstract forwarding model components.

Unlike OpenFlow, which analyzes packets based on a well-known set of protocols, P4 specifies parsing in a protocol-independent, programmable fashion. P4 supports parsing all the bytes of the packet and programmatically extracting specified headers. Based on the extracted headers, P4 processes the packet through ingress and egress pipelines, comprising a series of table match-action pairs that support conditional statements. Subsequently, the deparser constructs an output packet, and forwarding instructions, based on the table actions. Once a program specifies the abstract forwarding model, a compiler creates a binary for the specific switch implementation, including a Table Dependency Graph (TDG) allowing independent tables to be processed in parallel for speedy completion.

P4 provides sufficient flexibility to implement stateless FZS per subsection VI-B, along with stateless flow forwarding per subsection VI-E.

C. P4 IMPLEMENTATION

We developed a P4 implementation of the stateless FZS of subsection VI-C, including the stateless flow forwarding of subsection VI-E. At startup, each switch is configured with three parameters describing its location in network. These parameters can be configured using any convenient method, such as the stateless flow-zone address assignment process of subsection VI-F. The three parameters are:

- *switch type*: one of the three values *ToR*, *fabric*, or *spine*, indicating the node type, per subsection V-C
- *ZoneID1* and *ZoneID2*: two parameters identifying the switch zone in the network, assigned as in Fig. 2 for the three switch types. In particular:
  - if *switch type* = *ToR* then *ZoneID1* = *Pod ID* and *ZoneID2* = *Rack ID*
  - if *switch type* = *fabric* then *ZoneID1* = *Spine ID* and *ZoneID2* = *Pod ID*
  - if *switch type* = *spine* then *ZoneID1* = *Spine ID* and *ZoneID2* = *Spine Switch ID*

The programmability offered by P4 allows us to specify the parsing of incoming packets, enabling the extraction of the flow-zone address fields depicted in Fig 2.

The implementation used for the experimentation stores, in a P4 header structure, each of the bytes of the DA and SA for further processing. The first byte of each address contains the flow-zone address header, which identifies the address format as one of the four types shown in (see Fig. 2). Here we consider only server-directed frames (i.e., with the DA formatted as in Fig. 2a).

The P4 program next makes a frame forwarding decision based on the flow-zone forwarding method described in section VI. During frame descent, the forwarding method of subsections VI-A and VI-B is applied. During ascent, our study applies a simple form of the forwarding method of subsections VI-A and VI-E in which forwarding decisions are independent of the fields extracted from the SA. Therefore, since the destination is a server, the DA has the format of Fig. 2a and the following are the relevant fields:

- *Pod ID* = 2nd byte of DA
- *Rack ID* = 3rd byte of DA
- *Server ID* = 4th byte of DA
- *Flow Level 1 ID* = 5th byte of DA
- *Flow Level 2 ID* = 6th byte of DA

Each of these fields is presumed to contain the literal identifier of a forwarding port. As seen in the following paragraph, each switch uses only a subset of these fields and need not parse or store the others.

The forwarding method depends on the *switch type*:

- *switch type=ToR*: If *Pod ID=ZoneID1* and *RackID=ZoneID2*, the (downward) output port is identified as the frame's *Server ID* value. Otherwise, the (upward) output port identified as the frame's *Flow Level 2 ID* value.
- *switch type=fabric*: If *Pod ID=ZoneID1*, the (downward) output port is identified as the frame's *Rack ID* value. Otherwise, the (upward) output port identified as the frame's *Flow Level 1 ID* value.
- *switch type=spine*: The (downward) output port is identified as the frame's *Pod ID* value.

Next, P4 deparses the frame, constructing an egress frame that, in this case, is a duplicate of the ingress frame. Finally, the switch forwards this egress frame to the selected output port, as literally extracted from the DA.

D. SINGLE-SWITCH VALIDATION

After implementing the FZS solution, we performed unit testing, comparing the performance of stateless FZS to a baseline switching method. Both methods were programmed in P4 in a single switch, built with the "behavioral-model"(bmv2[2]) reference P4 software switch. Both switches were configured to parse the ingress frame, identify an output port, deparse the frame, and forward the egress frame, which was identical to the ingress frame. The only difference between the two methods was in the determination of the forwarding port. The FZS method extracted it literally from a field in the DA. The baseline switch performed a table look up the DA, to identify the egress port. We populated the forwarding table of the baseline switch with 5000 entries as a reference value for the lookup delay (considering 5000 as the minimum number of servers in an "hyperscale" data center). [1]

The unit test validation testbed consisted of one bmv2 switch connected to two hosts running in a mininet environment. Firstly, we measured the throughput using TCP traffic generated by iPerf,[3] with results depicted in Fig. 5a. FZS achieved an average throughput of 963 Mbit/s, compared to 853 Mbit/s with the baseline table-based solution.

We also measured the round-trip time (RTT) in the same validation scenario using the Linux ping tool. The results are shown in Fig. 5b. The stateless FZS solution achieved a lower average RTT of 0.27 ms, compared to 0.29 ms when using the baseline table-based solution.

In these experiments, the link capacities were unlimited, and the switch operations were all conducted in software, rather than in custom hardware optimized for table matching. The simulation results are therefore not intended to simulate the performance of a real switch but only to ensure FZS operation and to compare it with an estimate of the baseline

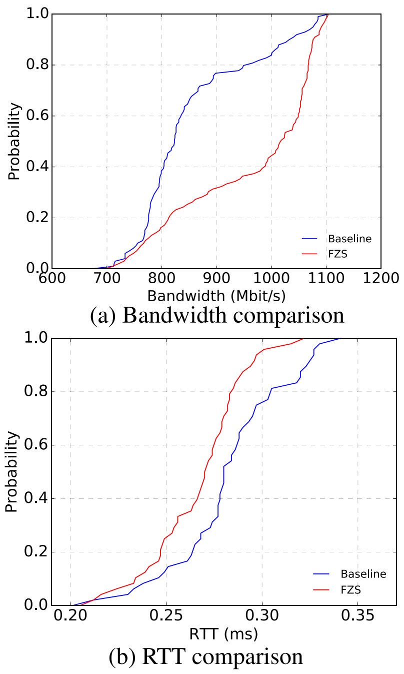


FIGURE 5. P4 switch comparison of baseline table-based switching vs. Stateless Flow-Zone Switching.

table-based switch performance. These results show that FZS performs better than the baseline table-based solution in the unit testing scenario, establishing a baseline result for the understanding of the experimentation performed in Section IX, regarding the performance of both software switching approaches on the machine used to perform the experimentation.

It is worth highlighting that in order to incorporate flow differentiation into the baseline table-based switch, we would need a more complex P4 pipeline, and a hash calculation to emulate Equal Cost Multi-Path (ECMP). We anticipate that this would add delay and reduce the overall performance. However, in the FZS case, this functionality is embedded directly in the flow level identifiers of the address and adds no processing burden.

With unit testing showing FZS performance better than the legacy approach, we proceeded to evaluation in a complete DCN.

IX. EXPERIMENTAL EVALUATION

A. SCENARIO

Herein we evaluate the FZS solution in a complete DCN. The experimental evaluation detailed in this section was conducted using the topology depicted in Fig. 6, which is consistent with that of Fig. 1 and Fig. 3. The evaluation topology comprises 4 spines of 4 spine switches each, 4 pods of 4 fabric switches and 4 rack switches each, and 2 servers per rack. The switches and servers are numbered in the figure in accordance with the location in the topology:

[2]https://github.com/p4lang/behavioral-model
[3]https://iPerf.fr

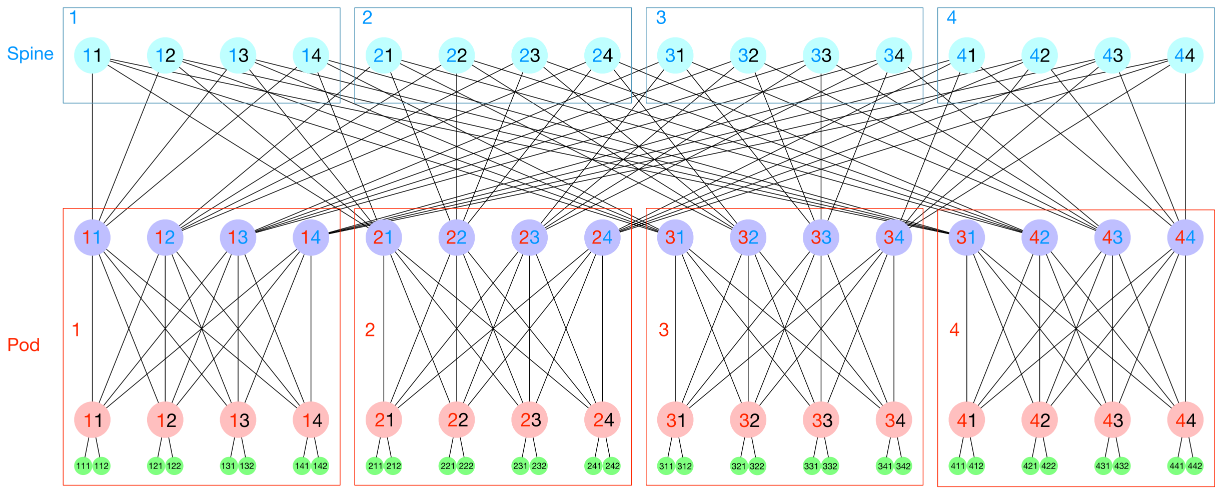


FIGURE 6. Data center topology used for the evaluation.

1) spine switches are labeled with a digit indicating the spine and a second indicating the spine switch within the spine;
2) fabric switches are labeled with a digit indicating the pod and a second indicating the spine;
3) ToR switches are labeled with a digit indicating the pod and a second indicating the rack;
4) servers are labeled with two digits indicating the rack and a third indicating server within the rack.

These identifiers are aligned with the element flow-zone address formats of Fig. 2.

The topology was created using Mininet,[4] a network emulator that uses process-based virtualization to run many hosts and switches on a single Operating System (OS) kernel. We set the link bandwidth to 10 Mbit/s. Each switch egress port was configured with 1.5 MB of buffer, per the default Linux configuration.

For TCP data, we ran iPerf (iPerf2) processes at each server. In iPerf terminology, the client process sends data to the iPerf server process. In our case, each server in the network executes one iPerf server process and three iPerf client processes. Each of the server's three iPerf client processes is configured to transmit TCP data blocks in a different data block size; the sizes are small (500 KB), medium (5 MB) and large (50 MB). Each iPerf client selects a server, opens a TCP connection to that server, transmits its data block, closes the TCP connection, and then repeats, beginning again by selecting a new server. Each connection is considered small, medium, or large, in accordance with the size of the transmitted data block. The servers are selected at random among those outside the pod of the client, so that all TCP traffic is forced through a spine switch.

All switches were programmed as described in subsection VIII-C, with *Flow Level 2 ID* identifying the spine and *Flow Level 1 ID* identifying the spine switch. Note that, in this example, the flow identification fields were not used as VM identifiers, but they could be used independently for that purpose as well with suitable partitioning.

A key feature of FZS is the flexibility to implement flow-dependent routing behaviors and, based on Software-Defined Addressing (SDA), specify the routing entirely at the end nodes, without reconfiguration of the network. To illustrate this potential, we implemented experiments using FZS to segregate TCP segments, routing elephants (data packets) and mice (ACKs) through different switches. This can illustrate not only the features of FZS but also its potential to implement new routing behaviors. Using a single experimental network configuration, as described above, we evaluate two different forwarding methods, based not on differences in the DCNs or the switch forwarding method but only on differences in the servers' creation of the frame DAs. The two methods are:

1) **Flow Distribution**: The servers create the DA *Flow Level 2 ID* and *Flow Level 1 ID* randomly, so that the ascending route to a spine switch is randomized over the whole topology. This method can be interpreted as a particular form of label-based segment routing, where the source of the packet indicates the destination and the route to reach it, although in this case the route is randomly selected by the source.
2) **Flow Segregation**: The servers again assign *Flow Level 1 ID* randomly, distributing both mice and elephant flows among the spine switches within the *Flow Level 2 ID* spine. However, mice and elephants are segregated into separate spines. The servers tag the DA of all ''elephant'' frames (TCP data frames) with DA *Flow Level 2 ID* selected randomly among the values 2, 3, or 4; they tag the DA of all ''mice'' frames (TCP ACK frames without data) with DA *Flow Level 2 ID* equal to 1. As a result, spine 1 is used exclusively for mice frames and spines 2-4 exclusively for

[4]http://mininet.org/

elephant frames. The intention of the segregation is that, by reserving switching capacity for mice and eliminating ACK competition with elephants in fabric and spine switches, we can reduce the ACK latency and its jitter and thereby improve network efficiency, in spite of the fact that the elephants will be limited to 75% of the spine capacity. This improvement may be estimated using a classic result (sometimes knows as the ''Mathis equation'' [21]) predicting the TCP throughput T as:

$$T < \frac{MSS}{RTT} \frac{C}{\sqrt{p}} \tag{1}$$

where C is a constant on the order of 1, MSS is the maximum segment size (typically 1460 bytes here), p is the packet (frame) loss rate, and RTT is the TCP round trip time. In this model, reducing RTT may improve the throughput, unless p increases drastically. In addition, reduced jitter may increase the accuracy of round-trip time estimates and thereby reduce the likelihood of false TCP timeouts, which result in unnecessary retransmissions and congestion window fallbacks.

In both methods, *Flow Level 2 ID* and *Flow Level 1 ID* are kept constant throughout the duration of a connection, routing all frames of a connection on the same path to prevent reordering.

B. STATIC LINK LOAD ANALYSIS

In order to better understand the scenario, we developed a static analysis of the network with an emphasis on understanding the extent to which its links are likely to be congested by data frames. As described above, the data frame sources select destination servers and spine routes randomly. Each such selection determines a full route. After each server selects a route for each of its three connections, we can analyze the links to determine how many connections each carries. We repeated this random selection to conduct a Monte Carlo simulation over many runs. The results are summarized in Table 2 and Table 3, using Flow Distribution and Flow Segregation, respectively. The tables refer to five link types:

- ''RF'' links go from a ToR switch to a fabric switch;
- ''FS'' links go from a fabric switch to a spine switch;
- ''SF'' links go from a spine switch to a fabric switch;
- ''FR'' links go from a fabric switch to a ToR switch;
- ''RS'' links go from a ToR switch to a server.

TABLE 2. Static Link Load using Flow Distribution.

Link Type:	RF	FS	SF	FR	RS
connection load per link: average	1.5	1.5	1.5	1.5	3.0
connection load per link: std. dev.	1.1	1.2	1.2	1.2	1.7
overloaded links	2.6	3.8	4.0	4.0	11.2
ratio of links overloaded	4%	6%	6%	6%	35%

TABLE 3. Static Link Load using Flow Segregation.

Link Type:	RF	FS	SF	FR	RS
connection load per link: average	2.0	2.0	2.0	2.0	3.0
connection load per link: std. dev.	0.6	0.7	0.7	0.7	1.7
overloaded links	4.9	6.5	6.8	6.8	11.2
ratio of links overloaded	10%	14%	14%	14%	35%

The RS load averages 3.0; an average of three connections is delivered to each server. Using Flow Distribution, each spine link type averages a load of 1.5 connections in each run, since the 96 connections are distributed across the 64 links at each level. Using Flow Segregation, that equivalent load is 2.0 in each run, since the data frame connections are distributed across only 48 links. The tables do not show the load of the first link, from a server to a ToR switch, because that is 3.0 in each case, since each server sends exactly three connections.

We also counted the ''overloaded links'', which we define to be those bearing more than the three connections borne by the server link to a ToR switch. If each such connection were delivered with one-third of the line rate, an overloaded link would be congested. Using Flow Distribution, approximately 4-6% of the spine links are overloaded. These could potentially result in congestion affecting not only data connections but also ACKs in mice flows. Using Flow Segregation, the ratio of overloaded spine links is more than doubled; however, this should not affect the delivery of mice flows, since they are routed through the reserved spine.

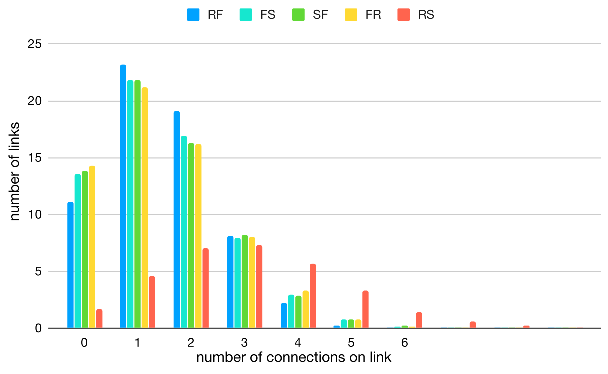
Additional detail is shown in Fig. 7, which provide histograms showing the number of links of each type for each level of connection load. As shown, some links are overloaded with more than four connections; this could lead to particularly heavy congestion. Note that RS link overload, which is independent of the network and routing, is dominant. As a result, the error rate p in Eq. (1) is expected to vary little between FD and FS, suggesting that improvement due to RTT may dominate.

C. EMULATION RESULTS

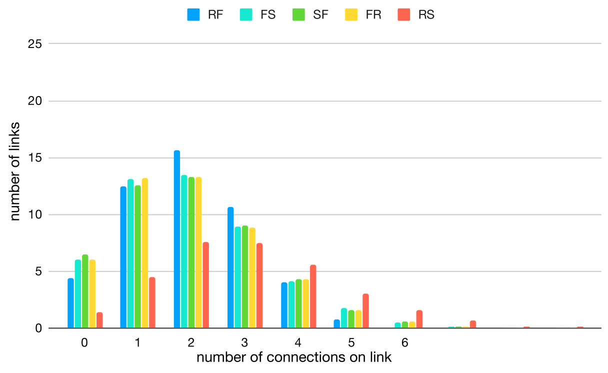
We analyzed the FZS DCN using the network and TCP traffic patterns discussed above, studying primarily:

- total transmission time to successfully transmit a 50 MB data chunk;
- TCP congestion window (which we abbreviate subsequently as ''Cwnd''), measuring the maximum Cwnd achieved per TCP connection;
- duplicated ACKs and retransmissions.

For each metric, we measured with both Flow Distribution and Flow Segregation, and we collected data individually for the three connection sizes. We also conducted each study with two common TCP congestion control algorithms: CUBIC and Reno. CUBIC, the default TCP algorithm in Linux

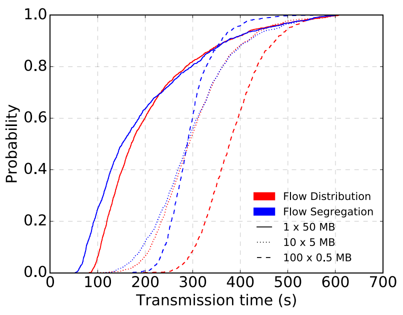


(a) using Flow Distribution

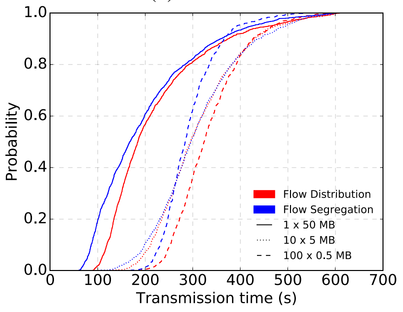


(b) using Flow Segregation

FIGURE 7. Link Load Histograms.

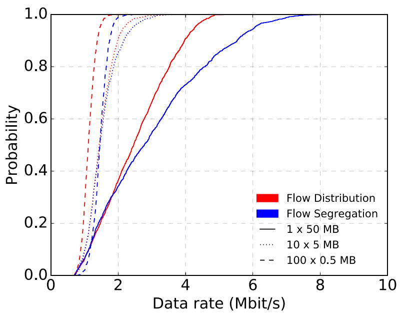


(a) CUBIC

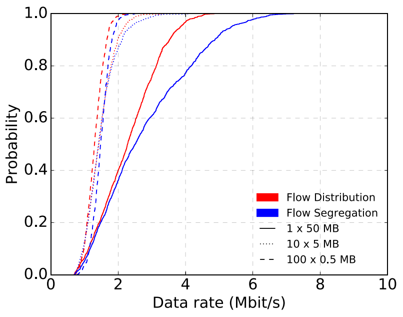


(b) Reno

FIGURE 8. Time to transmit 50 MB using different connection sizes.



(a) CUBIC



(b) Reno

FIGURE 9. Throughput transmitting 50 MB using different connection sizes.

kernels, uses a cubic function to grow the Cwnd. Reno grows the Cwnd using a linear function, resulting in a less aggressive increase than CUBIC. For both TCP variants, we used its native Linux stack implementation.

Fig. 8 presents cumulative distribution functions (CDFs) of the transmission time needed to convey a 50 MB data chunk in one large TCP connection, with a 50 MB data block; in 10 medium TCP connections, with 5 MB data blocks; and in 100 small TCP connections, with 500 KB data blocks. The results are averaged over all the iPerf clients during experiments of 600 seconds, repeated 30 times for both TCP variants and including only connections that completed within 600 s.

Fig. 9, derived from the data underlying Fig. 8, indicates the throughput of the chunk; we call this the ''chunk rate.'' As shown, only in a few cases did the chunk rate approach the maximum theoretical rate (the line rate of 10 Mbit/s minus about 5% in Layer 1–4 overhead).

To better understand the effect of Flow Segregation as enabled by FZS, Fig. 10 shows the maximum Cwnd achieved per connection, for the three connection sizes, as CDFs.

Using the data of Figs. 8, 9 and 10, we tabulated average results, in Tables 4 and 5.

These tables, summarizing Flow Distribution and Flow Segregation, respectively, present individual results for each of the three connection sizes and both of the TCP methods. The third column shows the chunk rate. The fourth column sums the three chunk rates in order to obtain a server data rate, totaled over all three connections transmitted by the server (excluding ACK transmission). The fifth column shows the average value of the maximum Cwnd attained during the course of each connection.

TABLE 4. **Results using flow distribution.**

Connection (MB)	TCP method	Chunk rate (Mbit/s)	Server rate (Mbit/s)	Max, Cwnd (MB)	reTx/ frame	Drop rate	Timeout rate
small (0.5)	CUBIC	1.13	5.12	0.25	0.1361	0.0659	0.0702
medium (5.0)	CUBIC	1.49		0.61	0.1072	0.0078	0.0993
large (50.0)	CUBIC	2.50		1.42	0.0936	0.0005	0.0931
small (0.5)	Reno	1.31	5.07	0.32	0.2166	0.0638	0.1528
medium (5.0)	Reno	1.46		0.60	0.1613	0.0079	0.1534
large (50.0)	Reno	2.30		0.92	0.0903	0.0005	0.0899

TABLE 5. **Results using Flow Segregation.**

Connection (MB)	TCP method	Chunk rate (Mbit/s)	Server rate (Mbit/s)	Max, Cwnd (MB)	reTx/ frame	Drop rate	Timeout rate
small (0.5)	CUBIC	1.46 (+29%)	6.01 (+17%)	0.25 (0%)	0.1016 (–25%)	0.0474 (–28.1%)	0.0542 (–23%)
medium (5.0)	CUBIC	1.53 (+3%)		0.62 (+2%)	0.1012 (–6%)	0.0069 (–12%)	0.0942 (–5%)
large (50.0)	CUBIC	3.02 (+21%)		1.79 (+26%)	0.0628 (–33%)	0.0003 (–40%)	0.0625 (–33%)
small (0.5)	Reno	1.48 (+13%)	5.76 (+14%)	0.32 (0%)	0.1624 (–25%)	0.0553 (–13%)	0.1071 (–30%)
medium (5.0)	Reno	1.49 (+2%)		0.60 (0%)	0.1471 (–9%)	0.0066 (–16%)	0.1405 (–8%)
large (50.0)	Reno	2.79 (+21%)		0.92 (0%)	0.0673 (–25%)	0.0004 (–15%)	0.0670 (–26%)

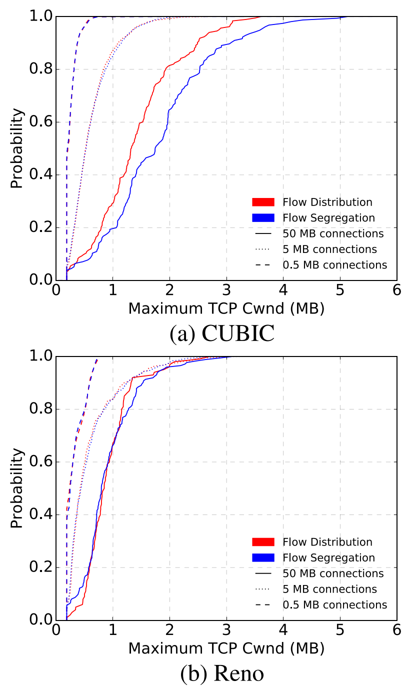


FIGURE 10. **Maximum TCP congestion window (Cwnd).**

Tables 4 and 5 also include data not related to the prior figures. In particular, the sixth column uses a TCP retransmission (reTx) count, as totaled over a sample interval of 300 s. The column displays the average reTx rate, as normalized to the number of frames sent during the sample interval; this was estimated based on the duration of the sample (300 s) and the average chunk rate, given in the third column, at 1500 bytes per frame.

In Tables 4 and 5, the seventh column shows the average unique duplicated ACK rate, again normalized to the number of data frames sent during the 300 s sample. Here, a unique duplicated ACK is a series of ACK frames without an increase in the acknowledgement number. Such a sequence indicates a dropped frame, a defective frame, or an out-of-order frame. Since out-of-order frames are essentially precluded by the network flow and defective frames unlikely given the experimental arrangement, we presume that each unique duplicated ACK indicates a dropped data frame, which, to our knowledge, can occur only as a result of buffer overflow. Therefore, we refer to this number as the "drop rate" of data frames, and we have labeled the column accordingly.

The TCP retransmissions represented in the sixth column are issued in response to both triple-duplicate ACKs (indicative of dropped data frames) and timeouts (indicative of dropped ACKs).

Therefore, the reTx rate minus the unique duplicated ACK rate should indicate the timeout rate, which is displayed in the eighth column and labeled accordingly.

Table 5 also displays the relative change, as a percentage, of the Flow Segregation results in comparison to the parallel Flow Distribution result of Table 4.

Based on these tables, we make the following observations regarding the data rates and Cwnd results:

1) Flow Segregation increases the server data rate of the network with both TCP methods, even though, as shown in the static link load analysis of IX-B, the connection load of the network links increases by 1/3 and the links are accordingly much more likely to be overloaded. In our study, the increase in server data rate was 17 % for CUBIC and 14 % for Reno.
2) The performance of CUBIC and Reno were similar using Flow Distribution. CUBIC benefited more from Flow Segregation, particularly for small connections, and performed better overall in that case.

3) In each case, the sum of the three maximum Cwnd values for each server was around 2-3 MB. This provides an upper bound on the average Cwnd from each server. Considering the 32 servers, this suggests a total Cwnd of considerably less than 100 MB at any given time. The Cwnd represents the data in flight, which is stored in buffers. The total amount of buffer space in the network (at 1.5 MB per switch egress port) is 432 MB. Consequently, it appears that the typical buffer was mostly empty, suggesting that the congestion was limited to a few switches at any given time. This matches our understanding of the network based on the static link load analysis of IX-B.
4) Flow Segregation had very little effect on the Cwnd of the small and medium connections and affected the Cwnd of large connections only using CUBIC. These results are more clearly illustrated by Fig. 10. It appears that the small connections may have been too short for TCP to reach a stable point. To illustrate, note that, for each small connection case, the average maximum Cwnd is at least half the connection size (e.g., for Reno, the Cwnd was 0.32 MB for a connection of 0.5 MB). This suggests that, at the time the maximum Cwnd was reached, half the total data was in flight, suggesting that the flow will complete in less than one round trip time afterwards. In spite of this ambiguous result concerning the window size, it is clear from Table 5 that the chunk data rate of the small connections was nevertheless improved by Flow Segregation, by near 29% using CUBIC. Therefore, it appears that the improvement may not be strictly explainable in terms of window size.
5) Flow Segregation had little effect on the maximum Cwnd of the medium connections; those might have also been too short for TCP to reach a stable point, with the maximum Cwnd about a tenth of the connection size. The chunk data rates of the medium connections were improved by Flow Segregation, but only slightly.
6) The chunk data rates of the large connections were significantly improved by Flow Segregation, but without a clear correlation to the maximum Cwnd. Other factors may better explain the improvements. For example, if Flow Segregation decreases the timeout rate, then the connections will spent more time operating near the maximum Cwnd point and less time shrinking the window and retransmitting.
7) Flow Segregation reduces the rate of retransmission due to both dropped data frames and timeouts, as seen in the last two columns of Table 5. For both TCP variants, the improvement in the retransmission rate correlates with the increased chunk rate, as particularly seen in the large and small connection data.
8) According to our basic understanding, the timeout rate is associated with dropped ACKs, resulting from buffers that are filled due to congestion. We expect Flow Segregation to alleviate ACK drops occurring in the spines but not to directly aid in drops at the ToR switches. Therefore, we anticipate Flow Segregation to significantly, but not entirely, reduce the timeout rate. This matches the results in Table 5.
9) The timeout rate is relatively independent of connection size. It is comparable to the drop rate for small connections but much larger for larger ones.
10) The drop rate is highest for the small connections and decreases for the medium and large connections, varying roughly with the inverse of the connection size. Based on this observation, we postulate that most of the packet drops occurs during the startup phase of the TCP stream; on aggregate, the small streams start up 10 times as often as the medium ones and 100 times as often as the large, which is consistent with the postulation and the data. Regarding the improvement that FS brings to the drop rate, in spite of the reduction in switching capacity for the elephant frames, we surmise that the improved ACK latency shortens the startup phase.

X. FOLLOWUP RESEARCH

FZS suggests many opportunities for future research. Here we note a few points.

1) **Access links:** Methods to reduce the bottleneck we observed at the access link from the server to the ToR switch should be considered. For example, multiple parallel links could be provided, and the role of flow identifiers in choosing an access link could be considered.
2) **Flow Routing:** Our evaluation considered two simple alternatives for flow routing: flow-neutral Flow Distribution, and Flow Segregation, which reserved a spine for TCP ACKs. Without changing the network, many other possible approaches are available simply by various forms of Software-Defined Addressing. Flows could be segregated in different switch configurations, such as by spine switch, and could be sorted based on different flow characteristics.
3) **Frames addressed to switches:** In V-D, we described flow-zone address formats suitable for frames addresses to the network switches, but our analysis was confined to frames addresses to servers. Many applications of FZS are relevant also to delivery of control-plane and management-plane frames to switches. For example, software-defined networking relies on timely communication among nodes and with controllers. As a direct extension of the methods described herein, we could consider a controller that would reprogram forwarding to select an alternative to the port specified in the frame, based on switch or link failures or temporary congestion.
4) **Additional use of flow identifiers by switches:** The FZS flow identifier could be used to separate flows into queues; for example, to segregate mice and elephants at the ToR switch and ensure that ACKs are on the fast lane, end to end. The flow identifier could be used

for other purposes, including, for example, exposing flow identifications directly at Layer 2 for flow control purposes.

5) **Additional use of flow identifiers by end nodes:** The FZS flow identifier could be used to encode header suppression fields to reduce overhead and possibly for security purposes. For example, a series of frames addressed from one VM to another may all refer to a common IP source address and a common IP destination address. In that case, the IP addresses could be suppressed in the network, where they are unneeded, and restored at the destination based on the flow identifier.
6) **Mobility:** A key aspect of the design of data center routing methodology is the need to accommodate VM mobility, since flexible usage demands the ability to seamlessly migrate VMs among servers. Traditionally, addresses serve to provide both an identity and a location, and mobility demands both. Historically, Layer 2 addresses have been flat and unique, characteristics suitable for identification, and Layer 3 addresses have been hierarchical, characteristics suitable for location. In the form of FZS described herein, the Layer 2 address serves as a complete identifier of the location of the server, and, if appropriate, VMs as well. In an typical data center, the VMs will also be assigned an IP address at Layer 3. In this case, the IP address serves as an identifier but does not necessarily provide location. As a result, the logical approach to migration is for the VMs to retain its identity; namely, its IP address, while the network updates the location (the Layer 2 address). Some sort of mapping from Layer 3 destination address to Layer 2 destination address is required. In the simplest case, FZS can use a form of ARP, preferably proxy ARP, for this purpose, or more efficient alternatives may be feasible.

XI. CONCLUSION

This work presents Flow-Zone Switching, a loop-free routing solution based on Software-Defined Addressing. The method embeds routing instructions directly in the MAC address, simplifying the switch hardware and processing requirements and eliminating the need for routing tables and lookups. FZS uses the octets of the MAC address to encode an address format identifier to identify the type of frame, two or three zone identifiers to locate the servers and switches in the typical topology, and one or more flow IDs to identify the traffic flows or flow types, allowing per-flow management with fine granularity, complex Quality of Service (QoS) management, and advanced resource allocation.

Key characteristics and advantages of this work can be summarized as follows:

- The FZS routing method minimizes the state in switches by embedding the routing instructions directly in the MAC addresses, completely removing the need for forwarding tables.
- The routing instructions embedded in the MAC address express the zonal location of the network element in the topology as well as flow identifiers allowing diverse flow management methods, differentiating FZS from the previous solutions.
- The approach does not incur in any overhead in the packet and it can be implemented so no distribution of control (e.g., label to path binding) is required.
- Each FZS frame contains the path towards destination and it specifies also the full path towards its source, without the intervention of any other protocol or mechanism.
- The FZS approach provides enhanced scalability through switch simplification and flexible MAC address semantic space.
- FZS can implement arbitrary traffic engineering mechanisms, even separating packets that, per traditional 5-tuple characterization, belong to the same flow (e.g., TCP data segment and TCP ACK).
- Our approach follows the latest innovations in IEEE 802 addressing, using the new features provided by IEEE 802c.
- As with the IETF RIFT[5] (Routing in Fat Trees) Working Group, the method is not generalized to all topologies but instead is designed to take advantage of the popular Fat Tree network topology.

To validate the above merits, we evaluated the performance of FZS in a four-spine Clos data center network structure, using two versions of TCP. We implemented two different routing methods entirely at the end servers, without reconfiguring the network in any way, simply by changing the Software-Defined Addressing approach. In the Flow Distribution (FD) method, flow addressing was used to distribute the flows randomly over the switching network. In the Flow Segregation (FS) method, we reserved one spine (25% of the network capacity) for ACKs, segregating the elephants (data frames) from the ''mice''(ACKs). The second approach exhibited better performance, with improved throughput that we attribute to improved ACK transmission more than offsetting the restricted switching fabric experienced by the ''elephant'' data frames.

We conclude that FZS is a robust routing method for data center networks, offering the opportunity for flexibility, higher performance, advanced QoS and traffic management capabilities compared to legacy solutions while reducing the network equipment cost by eliminating the need of routing tables.

REFERENCES

[1] C. Kidd. (Jul. 11, 2018). *What is a Hyperscale Data Center?*. [Online]. Available: https://www.bmc.com/blogs/hyperscale-data-center/

[2] A. Andreyev. (2014). *Introducing Data Center Fabric, The Next-Generation Facebook Data Center Network*. [Online]. Available: https://engineering.fb.com/production-engineering/introducing-data-center-fabric-the-next-generation-facebook-data-center-network/

[5]`https://tools.ietf.org/wg/rift/`

[3] 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," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 45, no. 4, pp. 51–62, Aug. 2009. [Online]. Available: http://ccr.sigcomm.org/online/files/p51.pdf

[4] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 4, pp. 63–74, Aug. 2008.

[5] A. Greenberg, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta, "Towards a next generation data center architecture: Scalability and commoditization," in *Proc. ACM Workshop Program. Routers Extensible Services Tomorrow (PRESTO)*, 2008, pp. 57–62.

[6] R. N. 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," *ACM SIGCOMM Conf. Data Commun.*, vol. 39, no. 4, pp. 39–50, Aug. 2009.

[7] E. Rojas and G. Ibáñez, "Torii-HLMAC: A distributed, fault-tolerant, zero configuration fat tree data center architecture with multiple tree-based addressing and forwarding," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Dec. 2012, pp. 2523–2528.

[8] E. Rojas, J. Alvarez-Horcajo, I. Martinez-Yelmo, J. M. Arco, and J. A. Carral, "GA3: Scalable, distributed address assignment for dynamic data center networks," *Ann. Telecommun.*, vol. 72, no. 11, pp. 693–702, 2017.

[9] D. Comer, R. H. Karandikar, and A. Rastegarnia, "DCnet: A new data center network architecture," in *Proc. IEEE 7th Annu. Comput. Commun. Workshop Conf. (CCWC)*, Jan. 2017, pp. 1–6.

[10] Y. Hu, M. Zhu, Y. Xia, K. Chen, and Y. Luo, "GARDEN: Generic addressing and routing for data center networks," in *Proc. IEEE 5th Int. Conf. Cloud Comput.*, Jun. 2012, pp. 107–114.

[11] F. Wang, L. Gao, S. Xiaozhe, H. Harai, and K. Fujikawa, "Towards reliable and lightweight source switching for datacenter networks," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, May 2017, pp. 1–9.

[12] D. Lopez-Pajares, J. Alvarez-Horcajo, E. Rojas, A. S. M. Asadujjaman, and I. Martinez-Yelmo, "Amaru: Plug&play resilient in-band control for SDN," *IEEE Access*, vol. 7, pp. 123202–123218, 2019.

[13] H. B. Acharya, J. Hamilton, and N. Shenoy, "From spanning trees to meshed trees," in *Proc. Int. Conf. Commun. Syst. Netw. (COMSNETS)*, Jan. 2020, pp. 391–395.

[14] *IEEE Standard for Local and Metropolitan Area Networks: Overview and Architecture*, Standard IEEE Std 802-2014. Jun. 30, 2014. [Online]. Available: https://ieeexplore.ieee.org/document/6847097

[15] *IEEE Standard for Local and Metropolitan Area Networks: Overview and Architecture–Amendment 2: Local Medium Access Control (MAC) Address Usage*, Standard IEEE Std 802c-2017. Aug. 25, 2017. [Online]. Available: https://ieeexplore.ieee.org/document/8016709

[16] *(Draft) Standard for Local and Metropolitan Area Networks: Multicast and Local Address Assignment*, Standard IEEE Project P802.1CQ, 2020. [Online]. Available: https://1.ieee802.org/tsn/802-1cq/

[17] R. Marks. (Jan. 24, 2018). *Address Assignment for Stateless Flow-Zone Switching in the Data Center*. IEEE Project P802.1CQ Contribution. [Online]. Available: http://www.ieee802.org/1/files/public/docs2018/cq-Marks-flow-zone-addressing-0118-v00.pdf

[18] *OpenFlow Switch Specification*, document ONF TS-025, Version 1.5.1, Open Networking Foundation, Mar. 2015.

[19] ONF. *Open Networking Foundation*. Accessed: May 11, 2020. [Online]. Available: https://www.opennetworking.org/

[20] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, and D. Walker, "P4: Programming protocol-independent packet processors," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 3, pp. 87–95, Jul. 2014.

[21] M. Mathis, J. Semke, J. Mahdavi, and T. Ott, "The macroscopic behavior of the TCP congestion avoidance algorithm," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 27, no. 3, pp. 67–82, Jul. 1997.

[22] K. Agarwal, C. Dixon, E. Rozner, and J. Carter, "Shadow MACs: Scalable label-switching for commodity Ethernet," in *Proc. 3rd Workshop Hot Topics Softw. Defined Netw.*, Aug. 2014, pp. 157–162.

[23] B. Balakiruthiga, P. Deepalakshmi, S. N. Mohanty, D. Gupta, P. P. Kumar, and K. Shankar, "Segment routing based energy aware routing for software defined data center," *Cognit. Syst. Res.*, vol. 64, pp. 146–163, Dec. 2020.

[24] M. Dave, "An efficient traffic management solution in data center networking using SDN," in *Proc. Int. Conf. Power Energy, Environ. Intell. Control (PEEIC)*, Apr. 2018, pp. 825–829.

[25] C. K. Dominicini, G. L. Vassoler, L. F. Meneses, R. S. Villaca, M. R. N. Ribeiro, and M. Martinello, "VirtPhy: Fully programmable NFV orchestration architecture for edge data centers," *IEEE Trans. Netw. Service Manage.*, vol. 14, no. 4, pp. 817–830, Dec. 2017.

[26] C. K. Dominicini, G. L. Vassoler, R. Valentim, R. S. Villaca, M. R. N. Ribeiro, M. Martinello, and E. Zambon, "KeySFC: Traffic steering using strict source routing for dynamic and efficient network orchestration," *Comput. Netw.*, vol. 167, Feb. 2020, Art. no. 106975.

[27] Q. Dong, J. Li, Y. Ma, and S. Han, "A path allocation method based on source routing in SDN traffic engineering," in *Proc. IEEE Int. Conf. Smart Cloud (SmartCloud)*, Dec. 2019, pp. 163–168.

[28] L. Fang, F. Chiussi, D. Bansal, V. Gill, T. Lin, J. Cox, and G. Ratterree, "Hierarchical SDN for the hyper-scale, hyper-elastic data center and cloud," in *Proc. 1st ACM SIGCOMM Symp. Softw. Defined Netw. Res.*, Jun. 2015, pp. 1–13.

[29] K. S. Ghuman, "Improving energy efficiency and bandwidth utilization in data center networks using segment routing," M.S. thesis, Dept. Elect. Comput. Eng., Univ. Ottawa, Ottawa, ON, Canada, 2017.

[30] K. S. Ghuman and A. Nayak, "Per-packet based energy aware segment routing approach for data center networks with SDN," in *Proc. 24th Int. Conf. Telecommun. (ICT)*, May 2017, pp. 1–6.

[31] C. Guo, G. Lu, H. J. Wang, S. Yang, C. Kong, P. Sun, W. Wu, and Y. Zhang, "SecondNet: A data center network virtualization architecture with bandwidth guarantees," in *Proc. 6th Int. Conf. (Co-NEXT)*, 2010, pp. 1–12.

[32] X. Jin, N. Farrington, and J. Rexford, "Your data center switch is trying too hard," in *Proc. Symp. SDN Res.*, Mar. 2016, pp. 1–6.

[33] S. A. Jyothi, M. Dong, and P. B. Godfrey, "Towards a flexible data center fabric with source routing," in *Proc. 1st ACM SIGCOMM Symp. Softw. Defined Netw. Res.*, Jun. 2015, pp. 1–8.

[34] C. Kim, M. Caesar, and J. Rexford, "Floodless in SEATTLE: A scalable Ethernet architecture for large enterprises," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 4, pp. 3–14, 2008.

[35] M. Martinello, M. Ribeiro, R. E. de Oliveira, and R. de Angelis Vitoi, "KeyFlow: A prototype for evolving SDN toward core network fabrics," *IEEE Netw.*, vol. 28, no. 2, pp. 12–19, Mar. 2014.

[36] N. Medhi and D. K. Saikia, "OpenFlow-based scalable routing with hybrid addressing in data center networks," *IEEE Commun. Lett.*, vol. 21, no. 5, pp. 1047–1050, May 2017.

[37] O. Osamudiamen and C.-H. Lung, "Segment routing green spine switch management systems for data center networks," in *Proc. IEEE Conf. Dependable Secure Comput. (DSC)*, Dec. 2018, pp. 1–8.

[38] R. M. Ramos, M. Martinello, and C. E. Rothenberg, "SlickFlow: Resilient source routing in data center networks unlocked by OpenFlow," in *Proc. 38th Annu. IEEE Conf. Local Comput. Netw.*, Oct. 2013, pp. 606–613.

[39] Y. Ren, T.-M. Huang, K. C.-J. Lin, and Y.-C. Tseng, "On scalable service function chaining with O1 flowtable entries," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, Apr. 2018, pp. 702–710.

[40] Y. Ren, T.-H. Tsai, J.-C. Huang, C.-W. Wu, and Y.-C. Tseng, "FlowTable-free routing for data center networks: A software-defined approach," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Dec. 2017, pp. 1–6.

[41] A. Schwabe and H. Karl, "Using MAC addresses as efficient routing labels in data centers," in *Proc. 3rd Workshop Hot Topics Softw. Defined Netw.*, Aug. 2014, pp. 115–120.

[42] Y. Sun, M. Chen, B. Liu, and S. Mao, "FAR: A fault-avoidance routing method for data center networks with regular topology," in *Proc. Archit. Netw. Commun. Syst.*, Oct. 2013, pp. 181–189.

[43] H. Wessing, H. Christiansen, T. Fjelde, and L. Dittmann, "Novel scheme for packet forwarding without header modifications in optical networks," *J. Lightw. Technol.*, vol. 20, no. 8, p. 1277, 2002.

[44] W. Zang, Z. Jin, and J. Lan, "An SDN based fast rerouting mechanism for elephant flows in DCN," in *Proc. 8th IEEE Int. Conf. Softw. Eng. Service Sci. (ICSESS)*, Nov. 2017, pp. 363–366.

[45] A. Zhao, Z. Liu, J. Pan, and M. Liang, "A novel addressing and routing architecture for cloud-service datacenter networks," *IEEE Trans. Services Comput.*, early access, Oct. 8, 2019, doi: 10.1109/TSC.2019.2946164.

SERGIO GONZALEZ-DIAZ received the bachelor's, master's, and Ph.D. degrees in telematics engineering from the University Carlos III of Madrid, Spain, in 2015, 2017, and 2020, respectively. He has participated in several H2020 research projects funded by the European Commission, such as 5G-Crosshaul or 5G-CORAL. He currently works as a Researcher with Atos Spain, focusing his research on programmable networks and network virtualization, on which he has published several papers in international conferences and journals.

ROGER MARKS received the B.S. degree in physics from Princeton University, the M.E. degree in electrical engineering from the University of Utah, and the Ph.D. degree in applied physics from Yale University. He is currently with EthAirNet Associates. He is active in IEEE standardization and has participated in IEEE 802, since 1998, serving during that time on the IEEE 802 Executive Committee and as the Chair of the IEEE 802.16 Working Group. He has served as the Technical Editor for various IEEE standards, including IEEE Standard 802c-2017 and the ongoing P802.1CQ Project. He chairs the IEEE 802 ''Network Enhancement for the Next Decade'' Industry Connections Activity (Nendica) and is a member of the IEEE Registration Authority Committee. He was the 1995 recipient of the IEEE Technical Field Award.

ELISA ROJAS received the Ph.D. degree in information and communication technologies engineering from the University of Alcala, Spain, in 2013. As a Postdoctoral Researcher, she has worked in IMDEA Networks and, later on, as the CTO of Telcaria Ideas S. L. She has participated in diverse projects funded by the EC, such as FP7-NetIDE or H2020-SUPERFLUIDITY. She currently works as an Assistant Professor with the University of Alcala. Her current research interests include SDN, NFV, the IoT routing, high-performance Ethernet, and data center networks.

ANTONIO DE LA OLIVA (Member, IEEE) received the Telecommunications Engineering and Ph.D. degrees from the Universidad Carlos III Madrid (UC3M), Spain, in 2004 and 2008, respectively. He has been an Associate Professor with Universidad Carlos III Madrid, since then. He has published more than 30 articles on different networking areas. He is an active contributor to IEEE 802, where he has served as the Vice-Chair for IEEE 802.21b and the Technical Editor for IEEE 802.21d. He has also served as a Guest Editor for *IEEE Communications Magazine*.

ROBERT (BOB) GAZDA received the B.S. degree in electrical engineering from Drexel University, the M.S. degree in computer engineering from Villanova University, and the M.S.E. degree from Carnegie Mellon University. He is currently the Senior Director of the Wireless Networking Laboratory, InterDigital. He is an Accomplished Engineering Professional and a Technologist with more than 20 years of industry experience in wireless telecommunications, networking, and embedded systems. He is also with InterDigital. He is leading a team focused on next generation network architectures and edge networking. His research interests include wireless communications and networking, mobility, edge and distributed computing, and real-time systems.