Switchboard: A Middleware for Wide-Area Service Chaining

Abhigyan Sharma
AT&T Labs Research
abhigyan@research.att.com

Yoji Ozawa
Hitachi Ltd.
yoji.ozawa.zp@hitachi.com

Matti Hiltunen
AT&T Labs Research
hiltunen@research.att.com

Kaustubh Joshi
AT&T Labs Research
kaustubh@research.att.com

Richard Schlichting
U.S. Naval Academy
schlicht@usna.edu

Zhaoyu Gao
UMass Amherst
gaozy@cs.umass.edu

Abstract

Production networks are transitioning from the use of physical middleboxes to virtual network functions (VNFs), which makes it easy to construct highly-customized service chains of VNFs dynamically using software. *Wide-area* service chains are increasingly important given the emergence of heterogeneous execution platforms consisting of customer premise equipment (CPE), small edge cloud sites, and large centralized cloud data centers, since only part of the service chain can be deployed at the CPE and even the closest edge site may not always be able to process all the customers' traffic. Switchboard is a middleware for realizing and managing such an ecosystem of diverse VNFs and cloud platforms. It exploits principles from service-oriented architectures to treat VNFs as independent services, and provides a traffic routing platform shared by all VNFs. Moreover, Switchboard's global controller optimizes wide-area routes based on a holistic view of customer traffic as well as the resources available at VNFs and the underlying network. Switchboard globally optimized routes achieve up to 57% higher throughput and 49% lower latency than a distributed load balancing approach in a wide-area testbed. Its routing platform supports line-rate traffic with millions of concurrent flows.

CCS Concepts • **Networks → Middle boxes / network appliances**; **Traffic engineering algorithms**; **Wide area networks**; **Network architectures**; • **Computer systems organization → Cloud computing**.

Keywords service chains, network functions virtualization, cloud computing, traffic engineering

ACM Reference Format:
Abhigyan Sharma, Yoji Ozawa, Matti Hiltunen, Kaustubh Joshi, Richard Schlichting, and Zhaoyu Gao. 2019. Switchboard: A Middleware for Wide-Area Service Chaining. In *20th International Middleware Conference (Middleware '19), December 8–13, 2019, Davis, CA, USA.* ACM, Davis, CA, USA, 13 pages. https://doi.org/10.1145/3361525.3361555

1 Introduction

The core business of large tier-1 network providers such as AT&T revolves around the deployment of network services such as home broadband, enterprise VPN, and cellular service. These services are implemented by assembling selected *network functions* such as routers, gateways, firewalls, proxies, intrusion detection systems, DDoS scrubbers, and content caches into *service chains* to realize the requisite end-to-end semantics. While such service chains have traditionally been constructed using hardware middleboxes and static interconnections, next-generation network architectures based on *virtualized network functions* (VNFs) and a VNF-centric *network cloud* infrastructure offer the opportunity to create new service types with a faster roll-out at a lower cost, thereby benefiting both customers and network providers [2, 11, 22].

In this paper, we present *Switchboard*, a middleware system designed to realize this vision by supporting the dynamic construction of wide-area service chains in a way that can be optimized based on network topology, customer traffic, and other factors. Among other advantages, this system enables service chains to be customized for each customer, moving beyond today's static regime in which all customers in a pool (e.g., business Internet users) receive the same service features despite their diverse needs. For example, a logistics enterprise can add specialized network traffic analysis for its Internet-connected vehicles in response to an emerging security threat either by creating a new service chain or by instantly inserting a new VNF into an existing chain. Moreover, these custom services can be offered in a way that is agnostic both to the type of edge network and to user mobility, making the service available even as users move from home broadband to office Wifi to cellular.

Switchboard's goal is to facilitate the creation of wide-area service chains on an infrastructure that spans customer premise equipment (CPE) [4], small edge cloud sites, and large centralized data centers. These platforms vary dramatically in the resources they have available, and range from sites that are internal to ISPs [2], to sites operated by business customers, to clouds run by 3rd-party providers. Support for wide-area chaining across such heterogeneous execution platforms distinguishes Switchboard from current approaches that largely support chaining only within a single site or on a single cloud platform [19, 24, 29, 30, 37]. While a starting point, the ability to construct wide-area service chains is critical for production networks given that only part of a service chain can be deployed on a CPE and even the closest edge site may not always be able to process all the customers' traffic due to resource limitations. Moreover, the ability to build service chains dynamically with support for 3rd-party clouds provides the customization needed to build solutions such as those outlined above.

A key aspect of Switchboard is that its design exploits principles from service-oriented architectures (SOAs) [31] by treating each VNF and each edge network as a *platform service*. Each such service is managed independently, and has its own internal structure potentially consisting of multiple instances of the VNF or edge network across geo-distributed sites. This approach has many advantages.

It hides the complexity of the underlying cloud platform and the mechanics of running a VNF on that platform, and potentially enables a richer catalog of VNFs by allowing the inclusion of VNFs running on 3rd party sites. It also allows Switchboard to interface with the present-day edge networks as well as future edge networks (e.g., a content-centric edge [23]). In short, it partitions control of service chains in a way that enables Switchboard to focus on the core problem of dynamically chaining VNFs across the wide area in a scalable way.

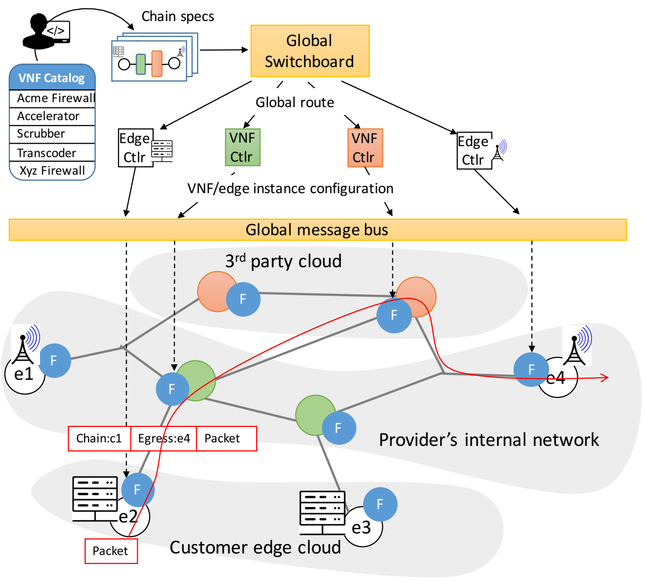


Figure 1. Switchboard control and data plane components. Circles labeled F are the Switchboard forwarders.

Switchboard's core functionality is realized by the following three middleware components (Figure 1).
1. Global Switchboard. A centralized controller that performs traffic engineering for wide-area chains using a scalable cost-based heuristic, and realizes chains by coordinating with VNF and edge services. Global Switchboard is in essence a Software Defined Networking (SDN) controller. (Section 4)
2. Switchboard forwarders. A proxy deployable on any cloud that can chain together VNFs running on a network provider's internal sites or 3rd party sites. Forwarders distribute connections among VNF instances using a hierarchical load balancing approach and support stateful VNFs by maintaining flow affinity and symmetric return paths. (Section 5)
3. Switchboard global message bus. A publish-subscribe mechanism that efficiently exchanges control plane state among Switchboard and other services. The message bus replicates control plane state in a fine-grained manner only at the required sites, thereby scaling to highly geo-distributed deployments. (Section 6)

We have prototyped and deployed Switchboard on private Openstack sites, 3rd party clouds, and ISP CPEs. Our prototype uses controllers based on OpenDaylight [27], a message bus based on ZeroMQ [49], a homegrown high performance DPDK-based forwarder [9], edge services based on OpenVPN and OpenFlow, and open-source VNFs including a caching proxy, a firewall, and a NAT.

We conduct extensive evaluation of Switchboard using prototype and network traffic simulations. In our experiments, Switchboard is responsive to dynamic events and can perform the addition of a new route or a new edge site to a service chain within a second. Switchboard's forwarders achieve throughput of 20 Mpps (equal to 80 Gbps for 500-byte packets) for 3 million concurrent flows using 6 CPU cores, and its global message bus achieves an order of magnitude lower latency than broadcast. In an analysis based on tier-1 network datasets, Switchboard's computationally efficient routing heuristic outperforms network-based distributed load balancing by an order of magnitude in throughput and provides latency within 8% of globally optimal routing across chains.

The paper is organized as follows. Section 2 shows how customers can use Switchboard to create service chains. Section 3 shows how Switchboard orchestrates service chain creation. Section 4 presents Global Switchboard network model and traffic engineering schemes. Section 5 discusses the proxy-based load balancing data plane and its safety and performance. Section 6 presents the topology of its message bus. Section 7 evaluates Switchboard and also compares it against alternatives. Section 8 discusses related work and Section 9 concludes.

2 Building service chains

Switchboard allows users to create custom network services by stitching together specified VNFs into a service chain. In this section, we provide a customer-centric view of Switchboard and demonstrate its wide-area service chaining capabilities across multiple cloud platforms.

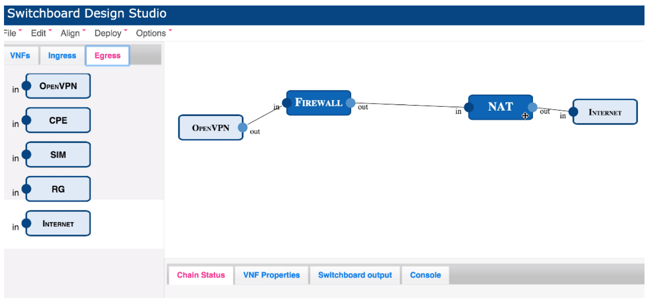


Figure 2. Switchboard customer portal showing creation of a service chain with a VPN as ingress, firewall and NAT as the VNFs in the chain, and Internet as the egress.

Figure 2 shows Switchboard's customer portal through its web interface. The portal displays the list of network functions available in Switchboard, which we envision evolving into an appstore-like marketplace where an ISP and 3rd-party vendors list VNFs that can be incorporated into customer service chains. In addition to selecting a VNF, a customer can define any customer-specific VNF configuration options through the portal, e.g., its firewall rules or caching policies. A customer does not need to specify deployment details such as cloud platform configuration, VNF location, and capacity, since those are automatic.

A customer defines the structure of a service chain by using the graphical user interface to specify its ingress, its egress, and the ordered set of VNFs in the chain. To define ingress and egress for a service chain, a customer specifies the type of edge network and the

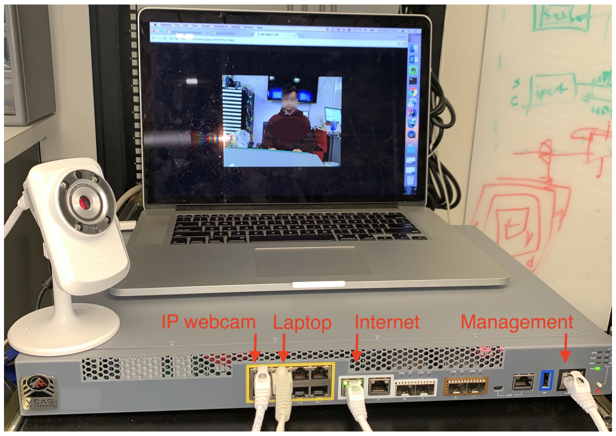


Figure 3. Demo of a face blurring VNF (running on AWS EC2) processing video stream from a customer premise equipment.

attributes specific to the edge network. For an enterprise customer with multiple office locations, for example, the ingress (similarly, egress) might be defined as the identifier for the customer edge router on one of its premises. If a service chain only applies to a slice of traffic at that location, this is specified using additional attributes such as VLAN IDs, or IP packet header fields such as source and destination IP prefixes and port ranges along with protocols.

The customer activates the chain through the portal, which results in automated route computation and installation. Upon completion, a status message is displayed to the customer. Thereafter, traffic between the customer-defined ingress and egress flows through the ordered VNFs specified in the chain in both directions.

To demonstrate Switchboard's capabilities, we built an example application that uses a VNF hosted at a remote site to perform custom video processing on a customer's traffic. Our testbed spans two locations, a local installation that represents a customer location and a remote installation in Amazon AWS EC2 [42]. The local installation consists of three physical devices: a customer-premise equipment box provided by the ISP (CPE), an IP-enabled webcam, and a laptop. The webcam and laptop are connected to the CPE via ethernet cables, and the CPE is connected to the internet over an ethernet cable. The CPE supports network functions executing in virtual machines, which we use to run Switchboard's data plane forwarder. The remote cloud consists of a video-processing network function and a Switchboard forwarder. The network function uses a GPU to perform face detection and to anonymize faces for privacy in real-time.

We define a service chain through the portal with the ingress as the subnet of the webcam, the egress as the subnet of the laptop, and the video-processing VNF as the only network function in the chain. Prior to chain activation, the default chain did not include any network functions. As a result, the Switchboard forwarder on the CPE routed traffic from the webcam to the laptop, where the original transmitted video could be viewed unmodified. After chain activation, the traffic now flows from the CPE to the network function at the remote site via the forwarders. The face-anonymized video stream is then viewed on the laptop as shown in Figure 3. We measured the end-to-end latency to be under a second, with most of the latency coming from the video processing at the network function. The rest of the forwarding and wide-area network transit typically adds only a few tens of milliseconds of latency.

3 Switchboard operational overview

Switchboard translates a customer's high-level specification of a service chain into data plane forwarding rules to connect VNFs deployed across geo-distributed sites.

Data plane operation. The first packet in a connection (Figure 1 enters at an ingress *edge instance*, which affixes two labels to it. The first label identifies the customer and its service chain, and the second label identifies the egress edge site. The labeled packet is received by a *Switchboard forwarder* attached to the ingress edge. The forwarder uses packet labels to send the packet to an instance (e.g. VM, container) of the first VNF in the chain, possibly at a different site. Upon receiving the packet, the VNF instance processes it and sends it to the adjoining forwarder. This sequence repeats until the packet reaches an egress edge instance, which removes its labels and sends it to the destination. Subsequent packets in the connection in the same direction are routed through the same instances, and those in the reverse direction of the connection are also routed through the same instances, but in the reverse order.

Realizing a service chain. Switchboard configures the data plane elements to realize the above path taken by a packet in the following three phases.

1. Prior to chain specification. Switchboard's service-oriented approach creates the necessary services—VNF, edge, and Switchboard itself—even before a chain is specified. A VNF service is a multi-site, multi-tenant service comprised of VNF instances at each site and a centralized *VNF controller*. An edge service, similarly, is comprised of edge instances and an *edge controller*. The Switchboard service is comprised of the Global Switchboard, a *local Switchboard* at each site, and forwarders at each site. The local Switchboard controls the horizontal scaling of forwarders at the site and performs aggregation of messages sent either by or to forwarders. The communication among these services is facilitated by a *global message bus* to which all data and control plane elements are attached. Using the message bus, VNF instances and edge controllers register themselves with Global Switchboard when added to the system, as well as with one of the forwarders at the site for sending their traffic.

2. Chain creation time. Service chain creation requires coordination across various Switchboard services as shown by the arrows in Figure 4. (1) Upon receiving the chain specification, Global Switchboard obtains ingress and egress sites for the chain from edge controllers. (2) Global Switchboard computes wide-area routes between ingress and egress sites and allocates unique labels to identify the chain and its wide-area routes. It uses a two-phase commit protocol to update the route and labels atomically at all edge and VNF controllers. Two-phase commit allows Global Switchboard to recompute the route if the proposed route is rejected by a VNF controller due to resource shortage. (3) It propagates these routes and labels to edge controllers, VNF controllers, and Local Switchboards. (4) Edge and VNF controllers allocate their instances at the sites in the wide-area route for this chain and publish their information on the global message bus. (5) Local Switchboards read edge and VNF information and combine it with the wide-area routes to compute load balancing rules, which are finally installed at Switchboard forwarders.

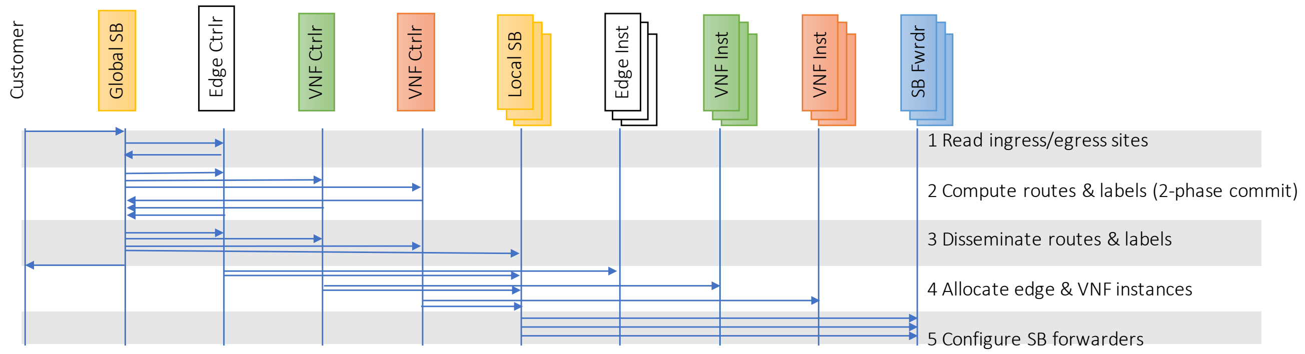

Customer
Global SB
Edge Ctrlr
VNF Ctrlr
VNF Ctrlr
Local SB
Edge Inst
VNF Inst
VNF Inst
SB Fwrdr
1 Read ingress/egress sites
2 Compute routes & labels (2-phase commit)
3 Disseminate routes & labels
4 Allocate edge & VNF instances
5 Configure SB forwarders

Figure 4. Flow of messages to configure a service chain route at chain creation time.

3. Connection setup time. The path of a connection's packets through VNF and forwarder instances is determined by Switchboard forwarders upon the arrival of the first packet of the connection. The load balancing rules at a forwarder specify a list of next-hop VNF or forwarder instances and their weights indexed by the service chain and egress site labels. A forwarder applies the weighted load balancing rule corresponding to labels on a packet to select a VNF or a forwarder instance for this connection. The instance selected for a flow is stored in a *flow table* entry for the connection keyed by the connection's labels and its header fields (specifically the 5-tuple of source IP, destination IP, protocol, source port, destination port), which allows later packets in the connection in the same direction to be routed through the same instance. A second flow table entry for the connection stores the previous-hop VNF or forwarder instance, which allows packets in the connection in the reverse direction to also be routed through the same VNF and forwarder instances in the reverse order.

4 Global Switchboard

Global Switchboard provides a platform on which traffic engineering algorithms can be implemented. To this end, it builds a model of the network based on data from various sources (Section 4.1), which enables multiple traffic engineering problems to be addressed (Section 4.2). It currently supports linear programming-based optimizations (Section 4.3) and an efficient dynamic programming algorithm for traffic engineering (Section 4.4) that are implemented in an open-source SDN controller (Section 4.5). A comparison of the performance of the linear programming and dynamic programming-based approaches appears in Section 7.3.

4.1 Network model

Switchboard models (Table 1) a set of network nodes N with a latency of $d_{n_1 n_2}$ between nodes n_1 and n_2. Cloud sites S are co-located with a subset of network nodes and have a compute capacity of m_s. A VNF f in the catalog F of VNFs is available at a subset of cloud sites S_f chosen by the VNF itself. The VNF also defines its compute capacity m_{sf} at each of those sites. The load per unit traffic on VNF $f \in F$ is denoted by l_f as reported by the VNF. Switchboard models the total load on a VNF at a site to be proportional to the total traffic sent or received by the VNF at that site.

The set of network chains defined by customers is C. The ingress and egress of chain $c \in C$ are $i_c \in N$ and $e_c \in N$ respectively, which are determined by the edge controller. Generalizations to multiple

Table 1. Parameters of Switchboard's network model.

Description	Notation
Network parameters	
Set of network nodes	N
Network delay from node n_1 to n_2	$d_{n_1 n_2}$
Frac. of traffic from loc. n_1 to n_2 on link e	$r_{n_1 n_2 e}$
Bandwidth of link e	b_e
Background traffic on link e	g_e
Maximum link utilization (MLU) limit	β
Cloud parameters	
Set of cloud sites	$S \subseteq N$
Max. allowed load at cloud site s	m_s
VNF parameters	
Set of VNFs	F
Set of cloud sites where VNF $f \in F$ is deployed	$S_f \subseteq S$
Max. allowed load of VNF f at cloud site s	m_{sf}
Load per unit forward traffic on VNF $f \in F_c$	l_f
Chain parameters	
Set of network chains	C
Ingress (egress) node of chain $c \in C$	$i_c\ (e_c) \in N$
Set of VNFs in chain $c \in C$	$F_c \subseteq F$
z-th VNF in chain $c \in C$	f_{cz}
Forward (reverse) traffic for chain $c \in C$ at stage z $(1 \le z \le (\lvert F_c \rvert + 1))$	$w_{cz}\ (v_{cz})$

ingress and egress nodes are omitted for ease of exposition. The ordered list of VNFs in chain C specified by a customer is denoted by F_c and the z-th VNF $(1 \le z \le (|F_c| + 1))$ in chain c is denoted by f_{cz}. A chain is represented as $(|F_c| + 2)$ nodes (including ingress and egress) and has $(|F_c| + 1)$ logical links between the nodes that we term *stages*. The forward (reverse) traffic for chain $c \in C$ at stage z $(1 \le z \le (|F_c| + 1))$ is denoted by w_{cz} (v_{cz}) and is obtained based on measurements by Switchboard forwarders for existing chains and on customer estimates for the initial chain deployment.

A network operator can leverage Switchboard for optimizing its network cost by specifying a constraint on the maximum link utilization (MLU) β – a commonly used cost function for traffic engineering [43]. In that case, Switchboard obtains the following additional inputs from the network operator: the set of links E in the network, the bandwidth b_e of each link $e \in E$, the network routing represented as the fraction $r_{n_1 n_2 e}$ of traffic between nodes n_1 and n_2 that crosses link e, and finally the background (non-Switchboard) traffic g_e on link e.

4.2 Traffic engineering problems

Chain routing. The routing for a chain is defined by the set of variables $x_{czn_1n_2}$, which denotes the fraction of traffic for a chain c at stage z from node $n_1 \in N_{cz}^{src}$ to $n_2 \in N_{cz}^{dst}$. This problem formulation implicitly assumes that the traffic from a chain at each stage can be split among multiple sites in arbitrary ratios, and that a VNF can allocate resources to a chain at a site at any granularity.

$$N_{cz}^{src} = \begin{cases} i_c & if\, z = 1 \\ S_{f_{c(z-1))}} & otherwise \end{cases} \tag{1}$$

$$N_{cz}^{dst} = \begin{cases} e_c & if\, z = (|F_c| + 1) \\ S_{f_{cz}} & otherwise \end{cases} \tag{2}$$

The objective of this problem is to minimize the aggregate latency of chains weighted by the fraction of traffic along each path for each chain as expressed below.

$$\sum_{c \in C} \sum_{z=1}^{|F_c|+1} \sum_{n_1 \in N_{cz}^{src}} \sum_{n_2 \in N_{cz}^{dst}} (w_{cz} + v_{cz}) d_{n_1n_2} x_{czn_1n_2} \tag{3}$$

VNF capacity planning. This problem seeks to provide hints to VNF providers for new deployment sites. In particular, given the number of new sites y_f for each VNF $f \in F$, the output of this problem is the set of sites S'_f that is non-overlapping with existing sites S_f and the capacities of VNFs at the sites in S'_f that minimizes the aggregate latency of chains as specified above.

Cloud capacity planning. Given additional cloud resources A to be deployed across all sites, this problem seeks to decide the resource a_s to be allocated to site S. Since capacity planning decisions are made to accommodate future increase in traffic demands, this problem seeks to maximize the factor of increase α over current traffic of each chain that can be supported, assuming traffic for all chains and all stages in a chain increases by the same factor.

4.3 Linear programming formulations

Chain routing. Switchboard's linear program solves the chain routing problem optimally. The problem lends itself to a linear program since the optimization objective (Equation 3) and the constraints are all linear. We present three key constraints of the problem here in the interest of brevity.

Compute. The total load across all chains and all VNFs at site $s \in S$ is less than its compute capacity m_s. A similar constraint bounds the total load on a VNF at a site to $m_s f$.

$$\begin{aligned} \sum_{c \in C} \sum_{f \in F_c : s \in S_f} l_f \Big(\sum_{s_1 \in N_{cz}^{src}} (w_{cz_{cf}} + v_{cz_{cf}}) x_{czs_1s} + \\ \sum_{s_1 \in N_{c(z+1)}^{dst}} (w_{c(z+1)_{cf}} + v_{c(z+1)_{cf}}) x_{czss_1} \Big) \le m_s \end{aligned} \tag{4}$$

Flow conservation. For chain $c \in C$, the fraction of traffic entering a VNF at stage z at any cloud site $s \in S$ equals the fraction of traffic at stage $(z + 1)$ that exits from it.

$$\sum_{n \in N_{cz}^{src}} x_{czns} = \sum_{n \in N_{c(z+1)}^{dst}} x_{c(z+1)sn} \tag{5}$$

Network cost. The total traffic on link $e \in E$ including background traffic and service chain traffic across all stages of all chains must be less than the MLU β.

$$g_e + \sum_{n_1 \in N} \sum_{n_2 \in N} r_{n_1n_2e} \Big(\sum_{c \in C} T_{cn_1n_2ce} \Big) \le \beta b_e \tag{6}$$

$T_{cn_1n_2}$ is a linear term and represents the traffic for chain c from node n_1 to n_2 across all stages of the chain combining traffic in forward and reverse directions.

$$\begin{aligned} T_{cn_1n_2} = & \sum_{z \in \{1,\cdots,(|F_c|+1)\} \,:\, (n_1 \in N_{cz}^{src}) \cap (n_2 \in N_{cz}^{dst})} w_{cz} x_{czn_1n_2} \\ + & \sum_{z \in \{1,\cdots,(|F_c|+1)\} \,:\, (n_2 \in N_{cz}^{src}) \cap (n_1 \in N_{cz}^{dst})} v_{cz} x_{czn_2n_1} \end{aligned} \tag{7}$$

VNF capacity planning. Switchboard uses a mixed integer program (MIP) adapted from the LP for the chain routing problem. This MIP introduces a new binary variable w_{fs}, the value of which determines whether the VNF $f \in F$ is placed at site $s \in S$ or not. The set of sites S_f where VNF f is deployed now includes all cloud sites S, since new VNF sites can include all cloud sites. The remainder of the formulation is updated accordingly. For instance, a new constraint specifies that unless a VNF $f \in F$ is deployed at a site, i.e., $w_{fs} = 1$, the corresponding chain routing variable cannot take a non-zero value.

Cloud capacity planning. We adapt the LP for the chain routing problem and makes the following key changes. First, the optimization objective is set to maximize the factor of increase α in chain traffic that can be sustained. Second, the capacity of each cloud site $s \in S$ becomes a variable $(m_s + a_s)$ instead of the constant m_s, with the constraint that the sum of additional capacity provisioned $(\sum_{s \in S} a_s)$ is less that A. Finally, the traffic for each chain is similarly represented as a variable, e.g., w_{cz} is replaced by αw_{cz}.

4.4 Dynamic programming algorithm

The dynamic programming algorithm computes routing for a single chain. It evaluates chain routes using a cost function, which is defined as the sum of latency cost, network utilization cost, and compute utilization cost. We denote the cost of the path from the z-th VNF in a chain (including ingress and egress edge VNFs) at site s to the $(z + 1)$-th VNF in the chain at site s' as $cost(s, z, s')$. The latency cost is numerically equal to the propagation delay from s to s'. Utilization-dependent costs are based on a piecewise-linear convex function that increases exponentially with utilization at values above 0.5 [15]. Network utilization cost is the weighted sum of costs of links that route traffic from s to s', weighted by the fraction of traffic carried on each link. Compute utilization cost is calculated from the utilization of the $(z + 1)$-th VNF at site s'.

For each pair of ingress and egress sites, the algorithm computes a list of sites such that the z-th VNF in the chain is deployed at the z-th site in the route. It constructs a table $E(z, s)$ whose entries denote the least cost for a chain route up to the z-th VNF in a chain while ending at site s. All entries in E are initialized to $+\infty$, except for the ingress edge site $E(0, ingress) = 0$. Entries for $z \ge 1$ are inductively computed as follows until the egress stage is reached.

$$E(z + 1, s) = \min_{s' \in S} E(z, s') + cost(s, z, s') \tag{8}$$

The least cost route is calculated by traversing E in the reverse direction starting from the least latency site for the last VNF in the chain. If the selected route can carry only a fraction of traffic for a

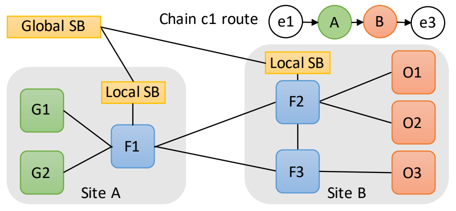


Figure 5. Forwarders running in VMs as an elastic service are incrementally deployable on any platform.

chain due to resource constraints, the algorithm is repeated to find the next least cost path to carry the remainder of the traffic until the routes for all the traffic for the chain is computed.

4.5 Implementation

Our prototype implements Global Switchboard as well as other controllers—Local Switchboard, an edge controller, and some sample VNF controllers—as modules in the OpenDayLight (ODL) framework [27]. The parameters of the network model (Table 1) for Global Switchboard are defined using the YANG data modeling language and data entries are stored as JSON objects. We then implemented the controller logic that is triggered upon updates to data items, e.g., a new chain specification initiates the process of deploying a chain. The linear programming optimization is implemented using a Java wrapper to the CPLEX optimization suite [8]. Our implementation also integrates a Java-based message queue to exchange control messages [49]. We plan to support fault-tolerance of controllers using a replication recipe based on MUSIC, a resilient key-value store optimized for wide-area deployments [6].

5 Switchboard data plane

Switchboard's data plane consists of a pervasive deployment of forwarders at every site (Section 5.1) that implement a hierarchical load balancing solution (Section 5.2). It supports key safety properties to realize service chains (Section 5.3), while providing line rate performance using a few cores per server (Section 5.4).

5.1 Deployment

A key deployment challenge is integrating Switchboard's forwarders with multiple cloud platforms such as CPEs, ISP edge clouds and 3rd-party clouds. Forwarders such as those available in Openstack Neutron [28], OpenContrail [24], and E2 [30] are implemented as a part of a cloud infrastructure's networking service running at the hypervisor layer. Such an implementation is possible in a provider's internal sites such as on an ISP edge cloud usually with vendor support. But, an ISP does not control the features available in a 3rd-party cloud such as EC2 which makes it difficult to deploy a forwarder. Further, high-performance VNFs bypass the hypervisor using NIC virtualization techniques such as SR-IOV [38], which makes it difficult to integrate forwarders even on an ISP-controlled cloud. These constraints dictate a new approach to forwarding.

Switchboard forwarders provide a cloud platform-agnostic service that is deployable in standalone VMs. A VNF instance (e.g., green VNF G1 in Figure 5) updates its local routing table to assign one of the forwarders as the proxy gateway for its service chain traffic. The VNF and its forwarder are kept in the same layer-2 domain for this interconnection. A forwarder communicates with another forwarder (possibly in a different site) over tunnels. As more VNF instances are added at the site, the Local Switchboard scales the number of forwarders elastically to support those VNF instances. This approach does not require any integration between Switchboard and the cloud platform, and only a small routing configuration change at the VNFs. Further, it also provides service chaining of high-performance VNFs that use NIC virtualization.

5.2 Hierarchical load balancing

A forwarder installs three sets of weighted load balancing rules using a hierarchical load balancing approach. These include (1) rules for the VNF instances with which it is associated (e.g., for F1 to load balance among G1 and G2 in Figure 5), (2) rules for the forwarders adjoining the next VNF in the chain (e.g., for F1 to load balance among F2 and F3), and (2) rules for the forwarders adjoining the previous VNF in the chain (not shown). A forwarder defines weights by takes the product of site-level weights, i.e., the variable $x_{czn_1n_2}$ in traffic engineering, with the weight of a forwarder or a VNF instance at that site. A VNF instance publishes its weight on the message bus. Similarly, a forwarder publishes its weight based on the sum of the weights of the VNF instances with which it is associated, e.g., the weight of F2 stored at F1 is the sum of the weights of O1 and O2.

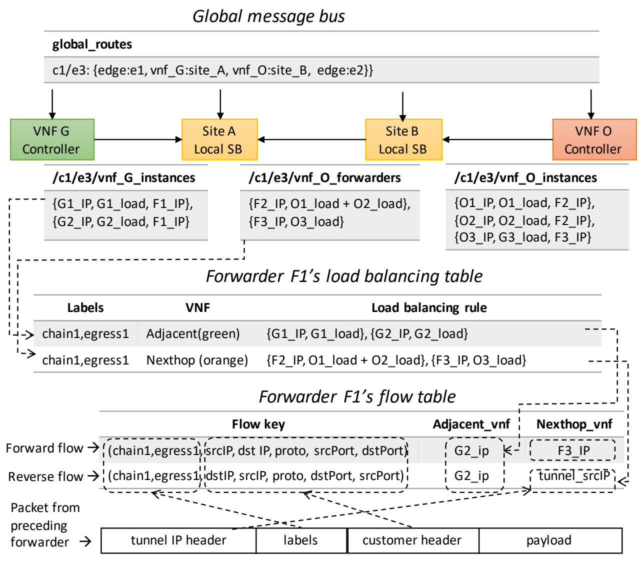

Global message bus
global_routes
c1/e3: {edge:e1, vnf_G:site_A, vnf_O:site_B, edge:e2}}
VNF G Controller
Site A Local SB
Site B Local SB
VNF O Controller
/c1/e3/vnf_G_instances
{G1_IP, G1_load, F1_IP}, {G2_IP, G2_load, F1_IP}
/c1/e3/vnf_O_forwarders
{F2_IP, O1_load + O2_load}, {F3_IP, O3_load}
/c1/e3/vnf_O_instances
{O1_IP, O1_load, F2_IP}, {O2_IP, O2_load, F2_IP}, {O3_IP, G3_load, F3_IP}
Forwarder F1's load balancing table
Labels
VNF
Load balancing rule
chain1,egress1
Adjacent(green)
{G1_IP, G1_load}, {G2_IP, G2_load}
chain1,egress1
Nexthop (orange)
{F2_IP, O1_load + O2_load}, {F3_IP, O3_load}
Forwarder F1's flow table
Flow key
Adjacent_vnf
Nexthop_vnf
Forward flow
(chain1,egress1)
(srcIP, dst IP, proto, srcPort, dstPort)
G2_ip
F3_IP
Reverse flow
(chain1,egress1)
(dstIP, srcIP, proto, dstPort, srcPort)
G2_ip
tunnel_srcIP
Packet from preceding forwarder
tunnel IP header
labels
customer header
payload

Figure 6. Flow table configuration at forwarders.

We demonstrate the sequence of messages to install a load balancing rule using Figure 6. To install rules for chain c1 at forwarder F1, Local Switchboard at the forwarder's site A receives the wide-area route for the chain c1 egressing at site e3. The route shows that the VNF G in the chain will be allocated at the same site A and the next VNF O will be allocated at site B. Local Switchboard infers that the forwarders at site A would need to load balance traffic among VNF G's instances at site A and then send that traffic to the forwarders associated with VNF O's instances at site B. Hence, it subscribes to the corresponding topics on the message bus—/c1/e3/vnf_G/site_A_instances and /c1/e3/vnf_O/site_B_forwarders—to receive the IP addresses and load balancing weights of VNF G's instances and VNF O's forwarders. These messages define the load balancing rules installed at F1.

5.3 Safety

A service chain should provide certain safety properties, as follows. *Conformity:* A customer's traffic must be routed through the specified sequence of VNFs. *Flow affinity:* All packets of a connection in a given direction are routed through the same sequence of VNF instances. *Symmetric return:* All packets of a connection in *both* directions are routed through the same VNF instances. The latter two properties are needed to support stateful VNFs; specifically, while some stateful VNF (e.g., traffic shapers) require only flow affinity, others (e.g., NATs) require symmetric return as well. Below, we discuss conditions under which these properties are supported and Switchboard's mechanisms to do so.

Conformity. Switchboard depends on labels applied by edge instances for conformity. An edge instance applies the first service chain label by parsing and matching the packet header fields to the chain specification. It applies the egress site label using a per-customer routing table that associates a destination address with an egress site. We note that existing route redistribution mechanisms to interconnect multi-site enterprise networks already maintain such routing tables, e.g., using Virtual Routing Functions [1, 25].

If a VNF supports Switchboard's labels, forwarders can correctly apply the load balancing rules to a packet even if that VNF modifies packet headers. Some VNFs may not support these labels as it may require non-trivial code changes [12]. Forwarders strip the labels before sending the packet to such VNFs. To re-affix the labels after the packet exits the VNF, forwarders must be able to uniquely associate the exit interface on the VNF with a set of labels. To this end, a VNF can provision separate instances for each set of labels. If the same VNF instance is shared among multiple sets of labels, the VNF must create separate interfaces for each of them.

Flow affinity and symmetric return. The flow table at a forwarder stores the load balancing selections made upon the arrival of the first packet in a connection. Figure 6 shows how the flow entries for the forward and reverse paths are populated. Forwarder F1's flow table entries store the adjacent VNF (G1), the next hop forwarder (F2), and the previous forwarder (not shown). Using these entries, later packets in a connection in any direction are routed to the same forwarder and VNF instances.

Forwarders maintain flow affinity and symmetric return despite changes in the wide-area route of the chain, addition or removal of VNF instances, or the load balancing weight for a VNF instance. To this end, forwarders allow existing entries in forwarders to remain until the completion of a flow and route only new flows on the new routes. Thus, existing connections continue to be routed to the same VNF instances in either direction.

Switchboard forwarders provide flow affinity if the VNF instances mapped to it maintain their association. However, elastic scaling or failure of a forwarder may remap a VNF instance to another forwarder, violating flow affinity. To safely change the VNF-to-forwarder mapping, flow table entries can be transferred across forwarders using recent proposals such as OpenNF [17]. We are developing a solution that supports elastic scaling and fault tolerance of forwarders by maintaining the flow table as a replicated distributed hash table across forwarder nodes. A discussion of the DHT-based forwarder is beyond the scope of this paper.

To support symmetric return globally, a flow egressing via an edge instance at a site must return to it in the reverse direction. Re-entry at a site is easily achieved for internal traffic between two customer locations. For Internet traffic to re-enter at the same site, the egress site should be the (unique) site from which the IP prefix of this customer at this site is advertised using BGP. Even if the reverse flow reaches the same site, it could be received at a different edge instance due to route asymmetry at the site itself. In these cases, we plan to use the above DHT-based flow table implementation to locate the original edge instance for the flow and route the traffic back to it to achieve symmetric return.

5.4 Performance

We evaluate the overhead of Switchboard's data plane using an OVS-based forwarder implementation and its throughput for millions of connections using a DPDK-based implementation.

OVS-based forwarder. The initial implementation of a Switchboard forwarder used Open vSwitch (OVS) [33], and in particular, the `multipath` and `learn` actions to implement the flow table. We performed an experiment to quantify the overhead of this forwarder over a simple bridge, and in particular, the overhead of maintaining flow affinity rules and additional overlay labels, i.e., MPLS labels for chain and route identification, and VXLAN tags for tunneling packets across the wide area. The overhead was measured by sending between 1 and 50 concurrent flows from one VNF instance to another via the forwarders.

Figure 7 shows the experimental setup and results. Compared to a normal bridge (c), we find that overlay labels (VXLAN+MPLS) add between 19-29% overhead (b), and flow affinity rules further add between 33-44% overhead (a). With more concurrent flows, the overhead reduces. VXLAN tunnels helps isolate Switchboard's traffic in a shared cloud while MPLS labels help it customize service chains. Flow affinity rules help maintain flow affinity for stateful VNFs. A service chaining solution that seeks to provide similar features as Switchboard is likely to incur similar overheads with OVS-based forwarding. However, a key limitation of the OVS-based forwarder is the poor scalability upon increasing the number of flows as shown in Figure 7, which led us to implement a significantly higher performance DPDK-based forwarder [9].

DPDK-based forwarder. We implemented and deployed a DPDK-based implementation of a Switchboard forwarder as a standalone VM on Amazon AWS, a private cloud, and a customer premise device (CPE) to provide wide-area service chaining. We then evaluated the performance of these forwarders in a scale-out deployment on SR-IOV-capable NICs. Our testbed is comprised of two servers directly connected using a 40 GbE cable, each having an Intel Xeon E5-2470 CPU (2.3GHz) and an Intel XL710 40GbE NIC. One of these servers hosts forwarder instances while the other hosts traffic generators and VNFs. We pin each Switchboard forwarder to a single core and assign a separate SR-IOV virtual interface to it. Each forwarder receives traffic from a traffic generator [10] and sends it to a unique VNF instance associated with the forwarder. We generate minimum sized (64B) UDP packets uniformly distributed among a fixed number of flows. We report the steady-state throughput as seen at the VNFs' interfaces in Figure 8.

The forwarder achieves a high throughput of up to 7 million pkts/sec (Mpps) with only a single CPU core (left-most bar in the figure). Each additional forwarder instance increases the throughput by 3-4 Mpps. Forwarders also show scalability with respect to

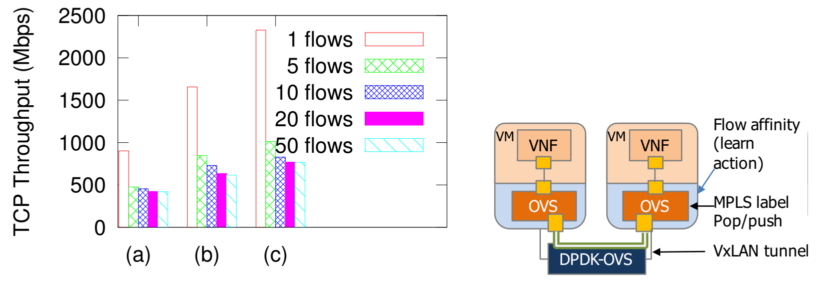


Figure 7. Compared to (c) a normal bridge, (b) overlay labels (VXLAN+MPLS) add 19% overhead and (a) flow affinity further adds 33% overhead for 50 flows.

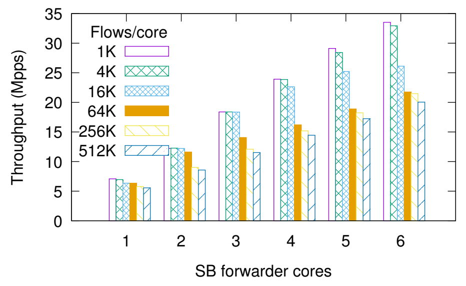


Figure 8. DPDK-based forwarder scales horizontally in throughput and num. of flows.

the number of concurrent connections. In the right-most bar in the figure, six forwarder instances store entries for a total of 3 million flows (= 512K×6) while still achieving an aggregate throughput of more than 20 Mpps. The throughput reduces with an increase in the number of flows due to lower CPU cache hit rates of flow table entries. In additional experiments with tens of millions of connection entries, we have found that once the flow table size far exceeds the CPU cache size, the throughput of a single forwarder core reaches a steady-state value in excess of 3 Mpps. The latency introduced by forwarders at the maximum throughput is 1 ms, but the latency at low to moderate loads is typically a few tens of microseconds. These latencies are consistent with other studies on similar hardware [45].

Extrapolating these results to a server with, for example, 32 cores, suggests that a Switchboard forwarder should be able to support nearly 100 Mpps, or 400 Gbps of forwarding capacity for an average packet size of 500 bytes. For a cloud site with say 400 Gbps of wide-area traffic, the cores available on a single server may be sufficient to support the throughput, connections and latency requirements of the service chaining data plane.

6 Global message bus

The global message bus implements a publish-subscribe topology with an optimized placement of subscription filters that outperforms broadcast-based techniques. The decoupling of publishers and subscribers fits our design, e.g., Global Switchboard does not need to know of changes in VNFs in a service chain, a forwarder attached to the VNF does not need to know the set of forwarders attached to the next or the previous VNF. To scale to hundreds of cloud sites, dissemination mechanisms should support policies to control which messages are sent to which sites. However, iBGP—the de facto tool for route dissemination in ISP networks—scales via hierarchical route reflectors [7], which takes away the ability to control which updates are sent to individual sites. Route reflectors must process all updates and could become dissemination bottlenecks, necessitating a new approach.

Switchboard's message bus topology plays a key role in determining the latency and the number of wide-area messages. Switchboard implements a message queuing proxy at each site to which all publishers and subscribers local to that site are connected. Publishers publish messages to its own site's proxy, and subscription filters are installed at the proxy on the *publisher's* site. The publisher's site is inferred from the topic itself. For example, the subscription of Site B's Local Switchboard to the topic /c1/e3/vnf_O/site_B_forwarders is installed in the proxy at site B. A site with no subscribers for a topic does not receive the message at all. A site that has any subscribers for a topic receives a single copy of the message over a TCP connection between the two proxies that is shared by all topics. The proxies at subscribers' sites send the message to their local subscribers. This approach results in the minimal number of messages being propagated across the wide-area and also a low propagation delay.

Switchboard's message bus helps extend service chains to new edge sites on-demand, thereby supporting use cases such as user mobility. When the the traffic from a service chain arrives at a new edge site, the message bus helps configure edge instances and their forwarders to route this traffic via the nearest existing wide-area route for the service chain. The message bus replicates wide-area routes and labels for all chains in Local Switchboard at every site. Based on the labels applied by the edge instance receiving a packet, Local Switchboard first chooses the route that results in the least latency to the egress site from this edge site. The message bus provides Local Switchboard with the list of forwarders assigned to the first VNF in the chain on the selected route. Finally, Local Switchboard configures forwarders to send traffic on that route towards the first VNF.

Comparison to broadcast. We compare the global message bus against a full-mesh broadcast for the setup shown in Figure 9. The testbed consists of VMs at a single site with emulated wide-area delays. Full-mesh sends a separate copy of a message for each subscriber whereas Switchboard only sends a single message for all subscribers at a site. Full-mesh results in excessive queuing of messages at the publisher's site, which results in an order of magnitude higher latency than Switchboard (Figure 9). Switchboard also has 57% higher throughput because full-mesh suffers from message drops due to buffer overflows. These results show that our message bus topology improves throughput and latency of control state dissemination by minimizing wide-area messages.

7 Experimental evaluation

In this section, we evaluate responsiveness of Switchboard's dynamic chaining capabilities (Section 7.1), perform end-to-end comparison of Switchboard to alternate service chaining approaches (Section 7.2), and evaluate Switchboard traffic engineering using a tier-1 network's dataset (Section 7.3).

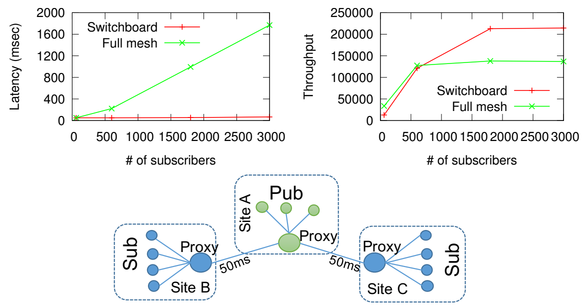
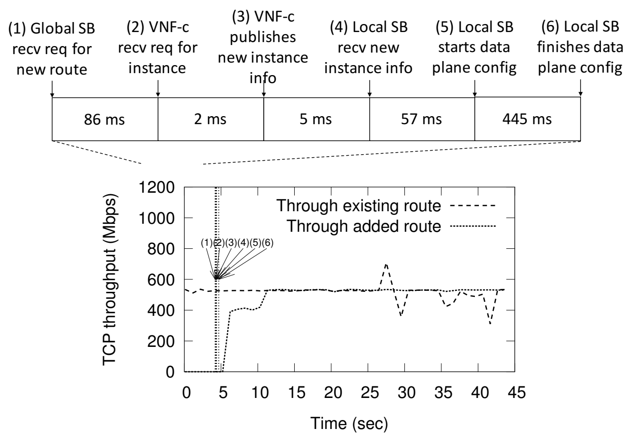


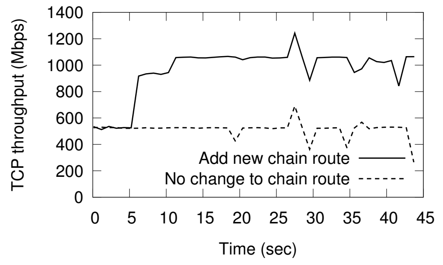
Figure 9. Message bus improves throughput by 57% and latency by more than 10× over a full mesh broadcast.

7.1 Dynamic service chaining

Chain route creation. We show Switchboard's ability to update chain routes dynamically and quantify the resulting performance improvements. Our VNF in this experiment is a NAT implemented using IP tables [35]. We performed this experiment at a single AWS EC2 site by creating two virtual sites A and B. Our service chain in this experiment has its ingress at site A and egress at site B. Initially, our service chain uses only a single NAT instance in site A. We manually trigger a new chain route creation by requesting Global Switchboard to create a new route via VNF instances in site B, which results in traffic being routed via VNF instances in both A and B.



(a) Event timeline during route update



(b) Comparison of total throughput

Figure 10

We highlight three findings from this experiment. First, as Figure 10a shows, a chain route update takes a total of only 595 ms. While an actual wide-area environment will increase the chain routing update time, our findings show that the overhead of Switchboard's control plane implementation is small. Second, in the same figure, the new chain route minimally affects the performance of the existing chain route and load is balanced evenly on the two routes. Third, in Figure 10b, the addition of a new chain route doubles the total throughput of the service chain compared to the case where there is no change in chain routing. The increase in throughput is commensurate to the additional capacity available on the new chain route. Thus, Switchboard enables VNFs to react to increases in load that overwhelm the resources in a single site by creating new routes via other sites.

Table 2. Latency in adding a new edge site to a chain.

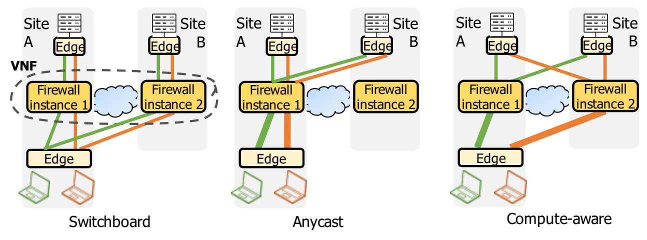
Operation	Latency
Local SB chooses the 1st VNF's site	0 ms
Edge instance's fwrdr receives 1st VNF's info	63 ms
Edge instance's fwrdr dataplane configured	93 ms
1st VNF's fwrdr receives edge's fwrdr info	74 ms
1st VNF's fwrdr starts dataplane configuration	233 ms
1st VNF's fwrdr finishes configuration	104 ms

Edge site addition. We evaluate the control plane latency of adding an edge site to a service chain. This scenario arises when a service chain user connects to a new edge site due to mobility, for example. Table 2 presents the latency of operations in configuring Switchboard's data plane to route traffic from the edge site to the first VNF in the chain (Section 6). The first three steps show the latency of configuring the forwarder at the edge site with load balancing rules and the tunnel to the first VNF's forwarder. The next three steps show the latency of configuring the first VNF's forwarder with the other end of the tunnel. The latency for the first step is 0 ms since it involves a simple computation at Local Switchboard itself. The total latency for the remaining operations is below 600 ms. This additional latency will be incurred only by the initial packet arriving at a new edge site. Hence, this experiment suggests that Switchboard can quickly add new edge sites to a chain to provide location-independent service chaining.

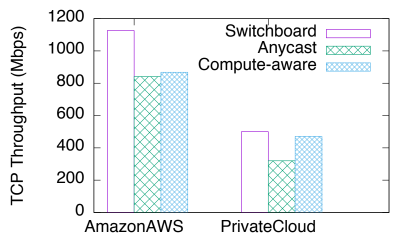
7.2 Switchboard vs. alternate service chaining approaches

E2E comparison vs. distributed approach. In this experiment, we show the importance of Switchboard's visibility across chains, VNFs, and sites in optimizing chain routing. We performed this experiment separately on Amazon AWS and on a private OpenStack-based cloud. On Amazon, we selected two sites with inter-site RTT of 150 ms. On the private cloud, we partitioned VMs into two virtual sites and emulated an inter-site RTT of 80 ms. These latencies are comparable to the maximum wide-area delays within the US. Our service chain, excluding the ingress and egress edge services, contains a stateful firewall VNF implemented using Linux IP tables. We specify two routes for this chain, with ingress and egress locations as shown in Figure 11a.

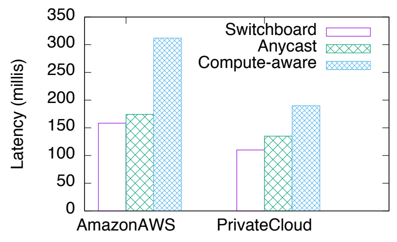
We compare Switchboard against two schemes based on distributed load balancing. Similar to anycast routing [14], Anycast



(a) Chain routing configurations of schemes



(b) TCP throughput comparison



(c) Latency comparison

Figure 11. Switchboard improves throughput by up to 57% and latency by up to 49% over distributed load balancing.

selects the site for the next VNF in a chain purely based on propagation latency, ignoring the available network link capacity on the route and the compute capacity available at that site. Compute-aware is similar to Anycast in that it considers sites in the order of lowest latency, but it does not pick a site if it does not have sufficient compute capacity.

In this experiment, all schemes have prior estimates of inter-site latency, a chain's traffic, and VNF capacities. In an actual deployment, these parameters are obtained from network operators tools [5], measurements at Switchboard forwarders, and reports by VNF controllers respectively. Anycast selects the firewall instance at site A for both chain routes in this experiment. Compute-aware selects the firewall instance at site A for the first chain route, which saturates the capacity of that VNF instance. Hence, it selects the firewall at site B for the second route. Switchboard computes routing via its LP-formulation to maximize throughput. Its computed routing distributes load among both instances and achieves the lowest propagation delays in the wide-area.

Figures 11b and 11c compare schemes based on the total TCP throughput and the average round trip latency between clients and servers on all routes. Switchboard's better load distribution across instances leads to 34% and 57% higher TCP throughput than Anycast. Further, Switchboard also achieves a lower latency than Anycast by 10% and 19% because Anycast has more traffic on its VNF instance in site A and consequently has longer queuing delays. Switchboard's lower wide-area propagation delays result in up to 49% and 43% lower latency compared to Compute-aware (Figure 11a). For the same reason, Switchboard also achieves a higher TCP throughput than Compute-aware by 39% and 7%. The throughput of Compute-aware relative to other schemes is worse on Amazon compared to the private cloud because of the higher latency and loss rates on wide-area links on Amazon.

While the quantitative improvements are dependent on the parameters such as VNF capacity, number of VNFs in a chain and wide-area latencies (see Table 1), these findings show the value of Switchboard's global optimization of chain routing over schemes that lack its visibility across chains, VNFs, and sites.

E2E comparison vs. unified approach. Switchboard's use of service-oriented design principles allows a VNF controller to share a VNF instance among multiple chains. In comparison, the unified controller approach (e.g., E2 [30] and Stratos [16]) vertically isolates a service chain by creating separate VNF instances for each chain. In this experiment, the VNF whose instances are shared across chains is the web cache Squid [39]. Squid intrinsically supports multi-tenancy by supporting multiple interfaces with custom caching rules. Our testbed spans two Amazon sites, with a 60ms RTT between them. We place clients and cache instances on one site, and the web servers hosting the content on the other site. We deploy five such service chains and evaluate two scenarios. The first uses a single cache instance for all chains and the second uses separate cache instances for each chain, each having one-fifth the size of the cache in the first case. Our workload follows a Zipf distribution with exponent = 1 and a mean file size of 50 KB.

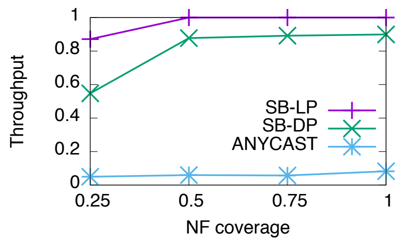
Table 3 compares the hit rate and the average download time for this experiment. We observe that sharing of a cache across five chains achieves 30% higher hit rate and 19% better download time compared to vertically siloed VNF instances. The explanation is that a shared cache enables reuse of cached objects across chains, thereby improving hit rates. In future work, we are investigating if other types of VNFs besides caches can benefit from sharing of instances across chains. We also expect shared VNF instances to lower provisioning costs because they can smooth out the variations in the traffic of individual chains by aggregation.

Table 3. Advantage of sharing a cache across chains.

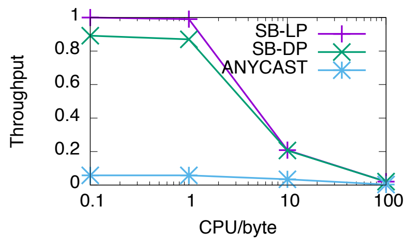
Scheme	Hit rate	Download time
Shared cache inst.	57.45%	56.49 ms
Vertically siloed cache inst.	44.25%	70.02 ms

7.3 Traffic engineering on tier-1 network datasets

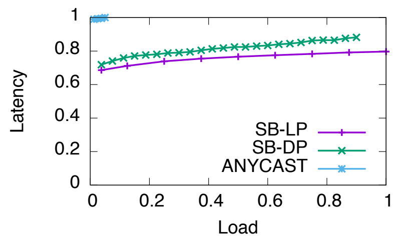
Simulation setup. We experiment with the backbone topology of a tier-1 network, which includes the link capacities and latencies, and the network routing. We assume that cloud sites have homogeneous capacity and are located near the nodes in the topology. Cloud sites run a total of 100 VNF services. A VNF is located at a fraction of all sites chosen randomly; we call this fraction the *coverage* of a VNF. At a site, capacity is divided equally among all VNF instances at that site. A VNF is modeled in terms of its compute cost per byte,



(a) NF coverage

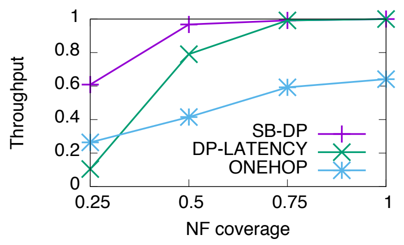


(b) CPU/byte

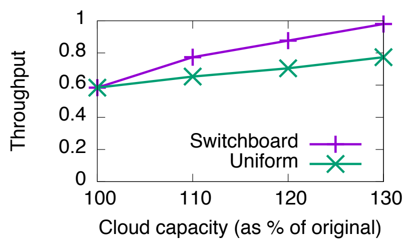


(c) Latency comparison

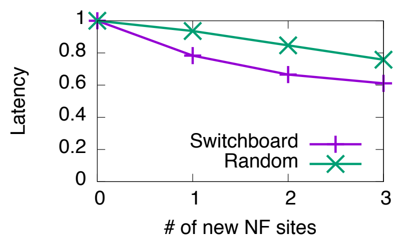
Figure 12. Switchboard's optimizations vastly outperform Anycast. SB-DP's performance is comparable to SB-DP.



(a) SB-DP microbenchmark



(b) Cloud capacity planning



(c) VNF capacity planning

Figure 13. (a) SB-DP's cost function and holistic route computation make it effective. (b, c) Switchboard helps improve cloud and VNF providers' baselines for deploying additional capacity.

denoted by *CPU/byte*. The compute load of a VNF is the product of CPU/byte and its traffic (in bytes/sec). We study the effect of coverage and CPU/byte in experiments below.

We use a snapshot of the tier-1 backbone traffic matrix collected in March 2015 to derive the traffic volume for each service chain. Network traffic is divided among Switchboard traffic and background traffic (e.g., transit traffic) in the ratio 4:1. Our workload consists of 10000 chains, whose source and destination nodes are chosen randomly. The traffic for a chain is proportional to the traffic at its ingress site. Each chain contains between 3 to 5 VNFs selected randomly. The sequence of VNFs in all chains is consistent with a pre-determined order of VNFs in agreement with typical sequences of VNFs in service chains, e.g., firewall is placed before a NAT.

Wide-area routing comparison. We compare the throughput achieved by the different schemes first. In Figure 12a, a higher NF coverage means that VNF instances are available at more locations, potentially alleviating wide-area network bottlenecks and improving throughput. A higher coverage does improve the throughput of Switchboard's routing schemes (SB-LP and SB-DP) that validates their network load-aware design. Anycast has more than an order of magnitude worse throughput than these schemes. It is unable to use a better NF coverage to improve its throughput because it always chooses the closest site hosting a VNF instance based on propagation delay, ignoring the network load on the path to that site or the available compute resources at that site.

In Figure 12b, low values of CPU/byte depict scenarios where the network is the bottleneck, whereas higher values of CPU/byte depict scenarios where the VNF's compute capacity is the bottleneck. Switchboard's schemes vastly outperform Anycast in all scenarios since Anycast is oblivious to either compute or network load. Surprisingly, SB-DP performs close to SB-LP with the difference being between 0%-11% (Figure 12a) and 11%-36% (Figure 12b), even though SB-LP's objective function in these experiments is to maximize its throughput.

Figure 12c evaluates latency for an increase in load assuming traffic for all chains increases by the same factor. Anycast cannot handle loads higher than 10% of the load sustained by SB-LP. Its latency is over 40% higher than SB-LP at low loads. Thus, Anycast's per-hop routing performs poorly on the latency metric as well.

SB-DP achieves a latency within 8% of SB-LP, even though SB-LP's objective function in this experiment is to minimize its latency. While SB-DP uses a simple, fast heuristic, SB-LP has much higher running time of up to 3 hours in this experiment. These and the above results suggest that SB-DP should perform well in practice and scale to larger topologies, and hence could be used as the primary routing scheme. However, SB-DP is a heuristic and may not work well in worst-case scenarios. Hence, SB-LP can run in the background at longer time scales than SB-DP and suggest better routes if such worst-case scenarios occur persistently.

To explain why SB-DP performs well, Figure 13a compares the throughput of two variants of SB-DP: DP-LATENCY and ONEHOP. DP-LATENCY uses only the propagation latency as its cost function. ONEHOP uses the same cost function as SB-DP, but it computes routes on a per-hop basis. SB-DP improves throughput by up to 6× and 2.3× over DP-LATENCY and ONEHOP respectively. While ONEHOP is consistently worse than SB-DP for any coverage, DP-LATENCY is worse than SB-DP when the coverage is less than 0.75. The key insight from this experiment is that both factors contribute to SB-DP's performance: its use of a cost function that considers latency, compute load, and network load, and its holistic computation of the entire service chain route.

Capacity planning. This experiment evaluates Switchboard's effectiveness in guiding cloud and VNF operators' decisions regarding capacity planning. In Figure 13b, Switchboard's algorithms for cloud capacity planning improve maximum throughput by up to 22% over the alternative of provisioning capacity uniformly across sites. In Figure 13c, we find that Switchboard's placement hints for deploying VNFs on new sites provide up to 27% lower latency than randomly selecting new sites.

Summary and future work. We find that Switchboard's global visibility across chains, VNFs, and sites improves service latency and throughput over decentralized schemes such as anycast and random site selection. A key insight is that our cost function for dynamic programming, which combines compute utilization, network utilization, and propagation delay using a simple heuristic, performs close to ideal on a real ISP topology and traffic dataset. In our future work, we plan to extend our network model to include time-varying traffic matrices and design routing algorithms for it. Another important area is to evaluate performance and cost metrics in case of network and compute failures. Finally, jointly optimizing Switchboard's traffic engineering and ISPs' wide-area traffic engineering is an interesting topic of future work [43].

8 Related work

Switchboard's novel features including wide-area service chaining and an SDN-controlled service-oriented architecture build upon a number of notable efforts in the following areas.

Service-oriented design. Switchboard is inspired by service-oriented architectures [31] for designing rich web services by combining standalone services based on protocols such as SOAP [40] that aid discovery and composition. More recently, the XOS [32] project for designing a cloud operating system shares our service-oriented approach. While XOS provides abstractions for layering and composition of services, we provide a concrete architecture for wide-area service chaining that explicitly defines the functionality of VNFs, edge services, and Switchboard in the platform. Distinct from a service-oriented design, our work proposes a holistic optimization of chain routing to construct service chains with low latency while increasing network throughput.

Service chaining. Several efforts provide mechanisms for service chaining. APLOMB [44] proposes outsourcing of enterprise middleboxes to cloud datacenters, focusing on a specific type of wide-area chaining between an enterprise site and a cloud datacenter. Other SDN-based solutions for service chaining have addressed the design of high-level Service Chaining APIs (ODL) [26] and compatibility with hardware OpenFlow switches (SIMPLE) [36]. In comparison, Switchboard seeks to construct arbitrary wide-area chains across any set of cloud sites and addresses the scalability of the control and data planes across a large number of sites.

ONAP [34] and its precursor ECOMP [3] are frameworks being developed by the telecom industry and AT&T respectively to design and orchestrate network services. As a general framework, ONAP would allow the implementation of the different Switchboard components as micro-services. For example, Switchboard's optimization algorithms could be implemented as micro-services in the OOF (ONAP Optimization Framework), ONAP SDN-C could be extended to support the configuration of the Switchboard forwarders, and ONAP DCAE could be used for data collection and analytics. Overall, Switchboard comes with a number of features to support wide-area chaining that goes far beyond the functionality of the current implementation of ONAP.

Segment Routing [13] and Network Services Headers [37] use source routing for service chaining. However, source routing can inflate packet header sizes, especially when using IPv6 headers or when routing though long chains of VNFs. In contrast, Switchboard's data plane uses label switching whose data plane overhead remains low even for longer chains.

DYSCO [48] supports dynamic reconfiguration of service chains (e.g., for VNF insertion) on ongoing connections without breaking them. Switchboard takes a simpler approach of only routing new connections via the reconfigured chains. Integrating DYSCO as part of forwarders would help us better support dynamic chaining.

Traffic engineering. Traffic engineering for service chains has seen prior work in optimization frameworks [20, 21] as well as routing algorithms [18, 41, 46]. In contrast to prior work, we pose new traffic engineering problems for service-oriented designs such as capacity planning and site selection for services. Also, we consider the interaction of overlay service chain routing with the underlying network routing, and propose a new dynamic programming algorithm for this problem that performs close to linear-programming based optimization on a tier-1 network dataset.

Switchboard can be compared to Google's Espresso [47]. Espresso is inter-domain traffic engineering solution that chooses an egress peering location and a network port at that location for each IP prefix but Switchboard computes overlay routes for chains of VNFs among sites managed by an ISP or on 3rd party clouds.

9 Conclusions

Service chaining is a critical capability in emerging network architectures based on VNFs, and one for which we contend no existing solution finds the right balance between VNF independence, dynamic control, and support for wide-area operation and multiple clouds. Switchboard is a middleware that alters this balance by splitting control between a global Switchboard controller and independent VNF controllers, thereby enabling holistic optimization across wide-area chains while supporting a rich ecosystem of VNFs. Switchboard's design rests on a scalable wide-area control plane in which control plane messages are efficiently propagated using a publish-subscribe event bus and a scale-out data plane that provides hierarchical, flow affinity preserving load balancing deployable on any cloud. Experimental results based on our ODL+OVS/DPDK-based prototype and tier-1 network datasets demonstrate the effectiveness of this approach in terms of latency, throughput, scalability and responsiveness of its components. Future work will focus on improving the traffic engineering aspects of the system and on continuing to gain practical experience with realistic applications.

Acknowledgements

We thank our shepherd Aniruddha Gokhale and the anonymous reviewers at Middleware for their excellent feedback that greatly improved the paper. At AT&T Research, we thank Hal Purdy and Brian Amento for their work on the face blurring demo and Scott Daniels for his implementation of the DPDK-based forwarder.

References

[1] Loa Andersson and Tove Madsen. Provider provisioned virtual private network (VPN) terminology, 2005.
[2] AT&T. AT&T Domain 2.0 Vision White Paper, 2013. https://www.att.com/Common/about_us/pdf/AT&T%20Domain%202.0%20Vision%20White%20Paper.pdf.
[3] AT&T. ECOMP (Enhanced Control, Orchestration, Management & Policy) Architecture White Paper, 2016. https://about.att.com/content/dam/snrdocs/ecomp.pdf.
[4] AT&T. AT&T Universal Customer Premises Equipment (uCPE), 2019. https://www.business.att.com/content/dam/attbusiness/briefs/universal-customer-premises-equipment-brief.pdf.
[5] AT&T. Global IP Network Home, 2019. https://ipnetwork.bgtmo.ip.att.net/pws/index.html.
[6] Bharath Balasubramanian. MUSIC-Multi-site State Coordination Service, 2018. https://wiki.onap.org/display/DW/MUSIC-Multi-site+State+Coordination+Service.
[7] Tony Bates, R Chandra, and E Chen. RFC 2796: BGP Route Reflection-An Alternative to Full Mesh IBGP. *The Internet Society, Network Working Group*, 2000.
[8] IBM ILOG Cplex. V12. 1: Users Manual for CPLEX. *International Business Machines Corporation*, 46(53):157, 2009.
[9] dpdk.org. DPDK: Data Plane Development Kit, 2016. http://dpdk.org.
[10] Paul Emmerich, Sebastian Gallenmüller, Daniel Raumer, Florian Wohlfart, and Georg Carle. Moongen: A scriptable high-speed packet generator. In *Proceedings of the 2015 Internet Measurement Conference*, pages 275–287. ACM, 2015.
[11] ETSI. NFV, 2016. http://www.etsi.org/technologies-clusters/technologies/nfv.
[12] Seyed Kaveh Fayazbakhsh, Luis Chiang, Vyas Sekar, Minlan Yu, and Jeffrey C Mogul. Enforcing network-wide policies in the presence of dynamic middlebox actions using flowtags. In *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*, pages 543–546, 2014.
[13] C Filsfils et al. Segment Routing Architecture, 2016. https://tools.ietf.org/html/draft-ietf-spring-segment-routing-04.
[14] Ashley Flavel, Pradeepkumar Mani, David Maltz, Nick Holt, Jie Liu, Yingying Chen, and Oleg Surmachev. Fastroute: A scalable load-aware anycast routing architecture for modern CDNs. In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*, pages 381–394, 2015.
[15] Bernard Fortz and Mikkel Thorup. Internet traffic engineering by optimizing OSPF weights. In *Proceedings IEEE INFOCOM 2000. Conference on Computer Communications.*, volume 2, pages 519–528. IEEE, 2000.
[16] Aaron Gember, Anand Krishnamurthy, Saul St John, Robert Grandl, Xiaoyang Gao, Ashok Anand, Theophilus Benson, Vyas Sekar, and Aditya Akella. Stratos: A network-aware orchestration layer for virtual middleboxes in clouds. *arXiv preprint arXiv:1305.0209*, 2013.
[17] Aaron Gember-Jacobson, Raajay Viswanathan, Chaithan Prakash, Robert Grandl, Junaid Khalid, Sourav Das, and Aditya Akella. OpenNF: Enabling innovation in network function control. In *ACM SIGCOMM Computer Communication Review*, volume 44, pages 163–174. ACM, 2014.
[18] Milad Ghaznavi, Nashid Shahriar, Shahin Kamali, Reaz Ahmed, and Raouf Boutaba. Distributed service function chaining. *IEEE Journal on Selected Areas in Communications*, 35:2479–2489, 2017.
[19] Sangjin Han, Keon Jang, Aurojit Panda, Shoumik Palkar, Dongsu Han, and Sylvia Ratnasamy. SoftNIC: A Software NIC to Augment Hardware. Technical Report UCB/EECS-2015-155, EECS Department, University of California, Berkeley, May 2015.
[20] Renaud Hartert, Stefano Vissicchio, Pierre Schaus, Olivier Bonaventure, Clarence Filsfils, Thomas Telkamp, and Pierre Francois. A declarative and expressive approach to control forwarding paths in carrier-grade networks. *ACM SIGCOMM computer communication review*, 45:15–28, 2015.
[21] Victor Heorhiadi, Michael K Reiter, and Vyas Sekar. Simplifying Software-Defined Network Optimization Using SOL. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)*, pages 223–237, 2016.
[22] Intel. Network Function Virtualization: Packet Processing Performance of Virtualized Platforms with Linux and Intel Architecture, Oct 2013. https://networkbuilders.intel.com/docs/network_builders_RA_NFV.pdf.
[23] Van Jacobson, Diana K Smetters, James D Thornton, Michael F Plass, Nicholas H Briggs, and Rebecca L Braynard. Networking named content. In *Proceedings of the 5th international conference on Emerging networking experiments and technologies*, pages 1–12. ACM, 2009.
[24] Juniper. Contrail, 2019. http://www.juniper.net/us/en/products-services/sdn/contrail/.
[25] David A Maltz, Geoffrey Xie, Jibin Zhan, Hui Zhang, Gísli Hjálmtỳsson, and Albert Greenberg. Routing design in operational networks: A look from the inside. In *ACM SIGCOMM Computer Communication Review*, volume 34, pages 27–40. ACM, 2004.
[26] OpenDaylight. OpenDaylight. Service Function Chaining, 2016. https://wiki.opendaylight.org/view/Service_Function_Chaining:Main.
[27] OpenDaylight. OpenDaylight Platform, 2019. https://www.opendaylight.org/.
[28] OpenStack. Neutron/ML2, 2019. https://wiki.openstack.org/wiki/Neutron/ML2.
[29] Openstack. Tacker, 2019. https://wiki.openstack.org/wiki/Tacker.
[30] Shoumik Palkar, Chang Lan, Sangjin Han, Keon Jang, Aurojit Panda, Sylvia Ratnasamy, Luigi Rizzo, and Scott Shenker. E2: a framework for NFV applications. In *Proceedings of the 25th Symposium on Operating Systems Principles*, pages 121–136. ACM, 2015.
[31] Randall Perrey and Mark Lycett. Service-oriented architecture. In *2003 Symposium on Applications and the Internet Workshops, 2003. Proceedings.*, pages 116–119. IEEE, 2003.
[32] Larry Peterson, Scott Baker, Marc De Leenheer, Andy Bavier, Sapan Bhatia, Mike Wawrzoniak, Jude Nelson, and John Hartman. XoS: An extensible cloud operating system. In *Proceedings of the 2nd International Workshop on Software-Defined Ecosystems*, pages 23–30. ACM, 2015.
[33] Ben Pfaff, Justin Pettit, Teemu Koponen, Ethan Jackson, Andy Zhou, Jarno Rajahalme, Jesse Gross, Alex Wang, Joe Stringer, Pravin Shelar, et al. The design and implementation of Open vSwitch. In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*, pages 117–130, 2015.
[34] Open Network Automation Platform. ONAP Architecture Overview Whitepaper (July 2019), 2019. https://www.onap.org/wp-content/uploads/sites/20/2019/07/ONAP_CaseSolution_Architecture_062519.pdf.
[35] Gregor N Purdy. *Linux iptables pocket reference.* O'Reilly Media, Inc., 2004.
[36] Zafar Ayyub Qazi, Cheng-Chun Tu, Luis Chiang, Rui Miao, Vyas Sekar, and Minlan Yu. SIMPLE-fying middlebox policy enforcement using SDN. In *ACM SIGCOMM computer communication review*, volume 43, pages 27–38. ACM, 2013.
[37] P Quinn et al. Network Service Header, 2013. https://tools.ietf.org/html/draft-quinn-nsh-00.
[38] Jack Regula. Multi-root sharing of single-root input/output virtualization, August 27 2013. US Patent 8,521,941.
[39] Kulbir Saini. *Squid Proxy Server 3.1: Beginner's Guide.* Packt Publishing Ltd, 2011.
[40] Kenn Scribner, Kennard Scribner, and Mark C Stiver. *Understanding Soap: Simple Object Access Protocol.* Sams, 2000.
[41] Vyas Sekar, Norbert Egi, Sylvia Ratnasamy, Michael K Reiter, and Guangyu Shi. Design and implementation of a consolidated middlebox architecture. In *Presented as part of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*, pages 323–336, 2012.
[42] Amazon Web Services. Amazon Web Services, 2019. https://aws.amazon.com.
[43] Abhigyan Sharma, Arun Venkataramani, and Ramesh K Sitaraman. Distributing content simplifies isp traffic engineering. In *ACM SIGMETRICS Performance Evaluation Review*, volume 41, pages 229–242. ACM, 2013.
[44] Justine Sherry, Shaddi Hasan, Colin Scott, Arvind Krishnamurthy, Sylvia Ratnasamy, and Vyas Sekar. Making middleboxes someone else's problem: network processing as a cloud service. *ACM SIGCOMM Computer Communication Review*, 42(4):13–24, 2012.
[45] VMWare. Deploying extremely latency-sensitive applications in vmware vsphere 5.5. https://www.vmware.com/content/dam/digitalmarketing/vmware/en/pdf/techpaper/latency-sensitive-perf-vsphere55-white-paper.pdf.
[46] Yanghao Xie, Zhixiang Liu, Sheng Wang, and Yuxiu Wang. Service function chaining resource allocation: A survey. *arXiv preprint arXiv:1608.00095*, 2016.
[47] Kok-Kiong Yap, Murtaza Motiwala, Jeremy Rahe, Steve Padgett, Matthew Holliman, Gary Baldus, Marcus Hines, Taeeun Kim, Ashok Narayanan, Ankur Jain, et al. Taking the edge off with espresso: Scale, reliability and programmability for global internet peering. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, pages 432–445. ACM, 2017.
[48] Pamela Zave, Ronaldo A Ferreira, Xuan Kelvin Zou, Masaharu Morimoto, and Jennifer Rexford. Dynamic service chaining with DYSCO. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, pages 57–70. ACM, 2017.
[49] Pure Java ZeroMQ. Pure Java ZeroMQ, 2019. https://github.com/zeromq/jeromq.