IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 25 May 2024, accepted 8 June 2024, date of publication 12 June 2024, date of current version 20 June 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3413016

RESEARCH ARTICLE

Exploiting SRv6 for Stateless and Per-Connection-Consistent Load Balancing

RYO NAKAMURA[1], KENTARO EBISAWA[2], HIDEAKI HAYASHI[3], TATSUYA FUJIWARA[3], AND TOMOKO OKUZAWA[4]

[1]Information Technology Center, The University of Tokyo, Tokyo 113-0033, Japan
[2]Independent Researcher, Kanagawa 211-0004, Japan
[3]Furukawa Network Solution Corporation, Kanagawa 254-0016, Japan
[4]Toyota Motor Corporation, Tokyo 100-0004, Japan

Corresponding author: Ryo Nakamura (upa@nc.u-tokyo.ac.jp)

ABSTRACT This paper proposes a new load-balancing method that supports Per-Connection Consistency (PCC) by leveraging ECMP with standardized routing protocols for operational simplicity. Load balancers must consistently deliver packets of a flow to one of the backend servers, even when the backend server pool changes. Achieving this property—PCC—is not a straightforward task and usually introduces additional complexity into networks, such as special packet forwarding mechanisms, synchronizing states across load balancers, and often complicated orchestration. Our approach, VRF-shadowing, achieves PCC on ECMP of routers without additional control plane systems by exploiting Segment Routing over IPv6 (SRv6). VRF-shadowing employs the daisy-chaining concept, achieving PCC in a stateless manner with a standardized SRv6-based Layer-3 VPN (L3VPN). VRF-shadowing is composed of two components: a minimal new feature for software of SRv6 routers and a new SRv6 End behavior at servers. We implemented the former in a commercial router and the latter in Linux and evaluated VRF-shadowing on an SRv6 L3VPN network built with only standardized routing protocols. The evaluation results demonstrate that VRF-shadowing preserves connections from being disrupted when the ECMP next-hops change. In the most severe scenario, VRF-shadowing reduced the failed HTTP requests rate from 77% to 0.7% compared to traditional ECMP.

INDEX TERMS Load balancing, per-connection consistency, segment routing, SRv6.

I. INTRODUCTION

Load balancing, which distributes application traffic among multiple servers, is a fundamental component of today's networked systems. There are various use cases of load balancing for dozens of servers [1], [2], large-scale data centers [3], [5], and cloud environments [6], [7]. Typical load balancers distinguish a connection, also known as a flow, by 5-tuples—source and destination IP addresses, port numbers, and IP protocol numbers. A load balancer receives traffic directed to an IP address representing a service (Virtual IP or VIP) and distributes connections to Direct IP addresses (DIPs) of backend servers associated with the service. Namely, load balancers calculate connection-to-DIP mappings and deliver connections using these mappings.

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

The simplest method to implement such a load balancer is to use Equal Cost Multi-path (ECMP). When a router has multiple equal-cost next-hops for a given destination prefix, it distributes the traffic destined for the prefix among the next-hops on a per-flow basis. ECMP functions as a load balancer when the destination prefix is a VIP, and its next-hops are DIPs. ECMP is a popular feature that is widely implemented in commercial off-the-shelf router products. Furthermore, operating ECMP does not require additional and/or dedicated control plane systems beyond traditional routing protocols. Thus, ECMP is the simplest and least expensive method for implementing load balancing in a network.

However, ECMP is known to have a major issue when used as a load balancer; it lacks Per-Connection Consistency (PCC) [8]. PCC is a load-balancing property that ensures a given connection is consistently delivered to the same

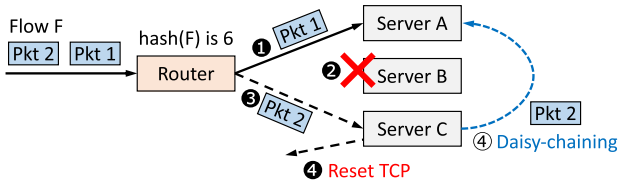


FIGURE 1. ECMP can violate PCC when it performs a load balancer. Next-hop changes can direct subsequent packets to different servers. Daisy-chaining is to save connections from disruption by redirecting stray packets to the correct servers (④) instead of resetting the connections (❹).

server from start to finish, even if the DIP pool changes. Figure 1 illustrates how ECMP can violate PCC. In the figure, a router forwards Flow F destined for a VIP. ❶ The router calculates a hash value for Flow F from its 5-tuple values and directs the packets of Flow F to Server A, chosen from three backend servers (equal-cost next-hops) based on the hash value. Then, ❷ if a server goes down and disappears from the next-hops on the router, ❸ the router directs subsequent packets of Flow F to Server C due to the change in the number of the next-hops. These subsequent *stray packets*, now directed to a different server, can disrupt the connections. ❹ Server C, not terminating the connection for Flow F, disrupts the connection by sending a TCP reset to the source. Thus, ECMP can violate PCC and disrupt connections when the next-hops change.

Dedicated load balancers are specifically designed to ensure PCC. The most common approach to achieving PCC is to enable load balancers to dynamically manage connection-to-DIP mapping entries, known as *stateful* load balancing. The size of the state can be as large as the number of connections; therefore, the data planes of stateful load balancers are typically implemented in software due to the need for substantial memory. An example is Google's Maglev [3], and there are several open-source software load balancers [9], [10], [11], [12]. In contrast to these software load balancers with relatively low throughput, several studies have explored implementing stateful load balancers in hardware by leveraging programmable hardware [4], [5], [13].

While mechanisms for distributing traffic have attracted attention, the control plane aspect is often overlooked from a practical standpoint. Load balancers require dedicated control plane systems for tasks such as configuring VIP-to-DIP mapping, detecting changes in DIP pools, and sometimes synchronizing the states of connection-to-DIP mapping. However, developing, operating, and maintaining such additional systems for load balancers can be challenging for modest-sized network operators. The paper [3] states that Maglev is a highly complex distributed system, and its implementation in production has faced numerous operational challenges. Unfortunately, not many can operate like hyper giants.

In contrast to load balancers that require dedicated (and often complex) control plane systems, we propose to exploit Segment Routing over IPv6 (SRv6) to enable ECMP to support PCC using only standardized routing protocols. Our approach, called **VRF-shadowing**, leverages *daisy-chaining* introduced by Beamer [14] and Faild [1]. In this concept, when a server receives a stray packet for which the server is not terminating the associated connection, it forwards the packet to the correct server selected for the flow in the previous state. For instance, Server C in Figure 1 forwards Packet 2 to Server A rather than sending a TCP reset to the source (④).

Previous load balancers based on daisy-chaining involve dedicated control and data planes to (1) manage the previous state of the mapping table in addition to the current state and (2) ensure that servers forward stray packets to the correct servers. We achieve both within the standardized SRv6 framework to democratize daisy-chaining-based load balancing for operational simplicity. VRF-shadowing implements the former through a separate routing table called **Shadow VRF**, which traces the previous state of the next-hops in another routing table. Shadow VRF is maintained by the router's CPU so that we can use the existing packet-forwarding hardware without modifications. The latter is accomplished by a new SRv6 End behavior, called **End.DT4.COND**, at the servers. End.DT4.COND redirects the received stray packets to Shadow VRF. The combination of Shadow VRF and End.DT4.COND provides PCC for ECMP over SRv6 Layer-3 VPN (L3VPN) with only standardized routing protocols.

This study inherits the basic concept from our previous work [15],[1] which implemented daisy-chaining over ECMP using a special packet-forwarding mechanism developed for programmable hardware switches. This study extends the concept and exploits SRv6 to leverage existing packet-forwarding planes. As new contributions, this paper describes how PCC is achieved within ECMP over SRv6 data and control planes (§III). Second, we implemented the Shadow VRF feature into a production SRv6 router, FITELnet FX2 from FURUKAWA ELECTRIC, and End.DT4.COND using extended Barkley Packet Filter (eBPF) [16] for Linux (§V). Third, we evaluate how VRF-shadowing ensures PCC during next-hop changes (§VI). In summary, the contributions of this paper include the following:

- We propose to exploit SRv6 to achieve load balancing based on daisy-chaining without dedicated control and data planes for operational simplicity.
- Our implementations demonstrate that minimal software modifications to SRv6 routers and a new SRv6 End behavior at the server network stack provide PCC for ECMP over SRv6.
- The proposed method was evaluated to its conceptual limits of daisy-chaining.

Our evaluation experiment demonstrates that VRF-shadowing achieves no connections are disrupted, whereas

[1]Our previous paper is written in Japanese, but the English version of the slides is available at https://github.com/ToyotaInfoTech/ecmper.

traditional ECMP disrupts 9.8% of connections due to next-hop changes. The evaluation also delves into the limitations of daisy-chaining in more severe scenarios. In the most severe case, VRF-shadowing significantly reduces the connection disruption rate from 77% to 0.7%. Daisy-chaining does not guarantee complete PCC. Nevertheless, it is a practical approach to prevent connection disruptions due to changes in the DIP pool [1], [2]. VRF-shadowing brings daisy-chaining to SRv6 networks, which transforms ECMP into PCC-aware, stateless hardware load balancers without the need for additional control plane systems.

II. RELATED WORK AND MOTIVATION

There are three primary design choices for load balancers: software implementation using per-flow mapping tables in the main memory, hardware implementation with tailored table structures for high throughput, and stateless hardware implementation without per-flow mapping. This section summarizes these approaches and describes our motivation to highlight the control plane aspect of load balancers from an operational perspective.

A. SOFTWARE LOAD BALANCERS

Implementing stateful load balancers in software is a straightforward approach for achieving PCC because it can store many connection-to-DIP entries in the main memory. Popular open-source software load balancers include Linux IPVS [9], HAProxy [12], Facebook Katran [10], and Github GLB [11]. Typically, the throughput of a single software load balancer is in the tens of Gbps range, which is less than that of the hardware load balancers.

To enhance the overall throughput, the deployment of low-throughput software load balancers requires scaling out. When deploying multiple software-based stateful load balancers, it is crucial that all load balancers consistently deliver a given connection to the same server; all load balancers in an operational domain need to synchronize the connection-to-DIP mapping table entries. Maglev [3] employs consistent hashing, enabling all load balancers to select the same DIPs for given connections without state synchronization. Another approach involves integrating load balancers with dedicated orchestration mechanisms, such as Aanata [6], specifically designed for and tightly integrated with Microsoft Azure's cloud infrastructure.

B. HARDWARE LOAD BALANCERS

Previous studies have explored implementing load balancers in hardware to achieve higher throughput than software implementations. Duet [7] utilizes ECMP of hardware layer-3 switches for load balancing, complemented by software load balancers for corner cases. However, relying on ECMP can violate PCC when updating the DIP pool. SilkRoad [4], on the other hand, implements a stateful load balancer in hardware using a programmable switch. Given the resource constraints of hardware switches, such as limited SRAM sizes, SilkRoad proposes a method to compress connection-to-DIP mapping entries and employs bloom filters to handle frequent changes in the DIP pool. Prism [13] proposes a hybrid approach that stores states only for connections whose DIPs are changed. Tiara [5] is also a hardware, stateful load-balancing architecture incorporating FPGA-based smart NICs and hardware programmable switches to handle large-scale traffic and over 10 million concurrent flows.

Stateless load balancers are well suited for hardware implementation because they do not maintain connection-to-DIP mapping entries. This characteristic also implies that it needs to preserve PCC without dynamic mapping. Beamer [14] solves this issue by introducing daisy-chaining. The Beamer load balancer divides the hash value space into buckets. Each bucket has a current DIP and a previous DIP associated with the bucket in the previous DIP pool state. When the DIP pool changes, the Beamer load balancer sends balanced packets with the previous DIPs embedded as an IP option. When a server receives a packet that does not match any established connections, it redirects the packet to the previous DIP embedded in the packet. Thus, daisy-chaining achieves PCC in a stateless manner.

Faild [1] is another stateless hardware load balancer that employs daisy-chaining to achieve PCC. Whereas Beamer embeds previous DIPs into the IP option, the Faild load balancer encodes previous DIP information into destination MAC addresses. This design allows Faild to be implemented with commodity-off-the-shelf layer-3 switches by statically manipulating ARP entries. Additionally, Cloudflare's Unimog [2], although a software load balancer, also uses the daisy-chaining approach to ensure PCC.

Cheetah [17] and SHELL [18] employ another approach, embedding cookies representing connection-to-DIP mapping information into specific fields in packets (e.g., Connection ID of QUIC, IPv6 addresses, and TCP Timestamp option). This design enables the Cheetah load balancer to always direct packets to the correct servers based on the cookies, thereby guaranteeing PCC. However, there is an issue of middle-box transparency due to the cookies.

C. MOTIVATION: THE CONTROL PLANE ASPECT

While the data plane aspect—how load balancers distribute and deliver packets—is crucial, the control plane aspect often receives less attention but is also indispensable for practical deployments. The responsibilities of control plane systems for load balancers typically include but are not limited to:

- Distributing VIP-to-DIP mappings across load balancers.
- Detecting changes in DIP states and updating DIP pools.
- Synchronizing connection states across load balancers.
- Calculating and deciding load distribution while collecting telemetry data.
- Detecting and responding to changes in other components, such as underlying routing.

Deploying load balancers involves operational costs for the load balancers themselves and requires significant efforts

to develop and maintain associated control plane systems responsible for the tasks mentioned above. In previous literature, a popular approach for control tasks is to assume some dedicated controllers [6], [7], [18], [19], [20]. Others say that configurations are read from files or received from external systems through RPC [3] or employ Zookeeper [14]. Leveraging commodity hardware switches involves running agents on the switches for control tasks [1], [7]. As such, network operators face additional complexity in network design and increased operational costs to manage load balancers.

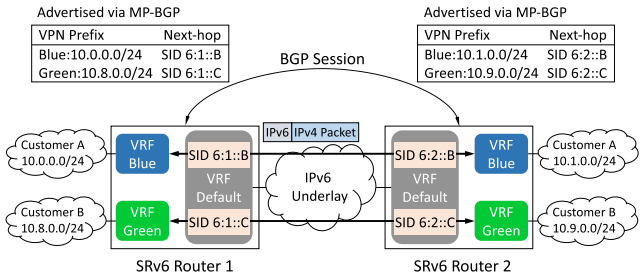


FIGURE 2. Overview of SRv6-based Layer-3 VPN.

III. APPROACH

This paper aims to achieve PCC without requiring dedicated control plane systems that often bring additional complexity. Our approach, called VRF-shadowing, employs the daisy-chaining concept introduced by Beamer and Faild—servers deliver stray packets to the correct servers. VRF-shadowing brings PCC to ECMP in SRv6 Layer-3 VPN with only standardized control planes of IGP and BGP for operational simplicity. This section first provides a brief overview of SRv6-based L3VPN and next describes VRF-shadowing.

A. SEGMENT ROUTING OVER IPV6 AND LAYER-3 VPN

Segment Routing (SR) [21] is a recent packet-forwarding paradigm that enables source routing. SR represents any topological entities as *segments*, e.g., links, nodes, and adjacency. A series of segments (segment list) embedded in a packet indicates where the packet should flow and how the packet is to be processed.

SRv6, a specific implementation of SR, uses IPv6 addresses as identifiers for segments (Segment Identifiers, or SIDs). The Segment Routing Header (SRH) [22], an IPv6 extension header, contains a segment list. SRv6 nodes interpret SRH and process packets according to the embedded SIDs. RFC8986 [23] defines the basic behaviors of SRv6 nodes. For example, End.DT4 decapsulates IPv4 packets encapsulated in SRv6 and submits them to an associated IPv4 routing table lookup. End.DT46, a variant of End.DT4, handles inner packets of both IPv4 and IPv6 according to their respective protocol families. SRv6 nodes have their SIDs (IPv6 addresses and associated behaviors) locally and execute these behaviors for packets destined for local SIDs. This concept—encoding how packets are to be processed as segments indicating specific behaviors—is called SRv6 Network Programming.

L3VPN is a popular use case of SRv6 [24]. Figure 2 illustrates an overview of L3VPN with SRv6 transport. Routers capable of L3VPN manage separate routing tables, called Virtual Routing and Forwarding (VRF), for each customer. In SRv6-based L3VPN, routers maintain local End.DT4 (and End.DT6, or End.DT46) SIDs for their local VRFs and advertise prefixes in the VRFs (VPN prefixes) along with the associated SIDs as next-hops via BGP. In Figure 2, Router 1 and Router 2 first exchange IPv6 addresses assigned to their loopback interfaces via IGP, then establish a BGP session with each other via the loopback addresses and subsequently exchange VPN prefixes through the BGP session. Consequently, Router 1 routes packets from Customer A's 10.0.0.0/24 network in VRF Blue to the 10.1.0.0/24 network in VRF Blue of Router 2, encapsulating the packets in IPv6 with a destination IPv6 address of 6:2::B, which is Router 2's End.DT4 SID for VRF Blue. In summary, SRv6-based L3VPN leverages IPv6 for transport and utilizes End.DT SIDs to identify the appropriate VRFs for inner packets at egress SRv6 nodes, similar to VPN route labels in MPLS VPN.

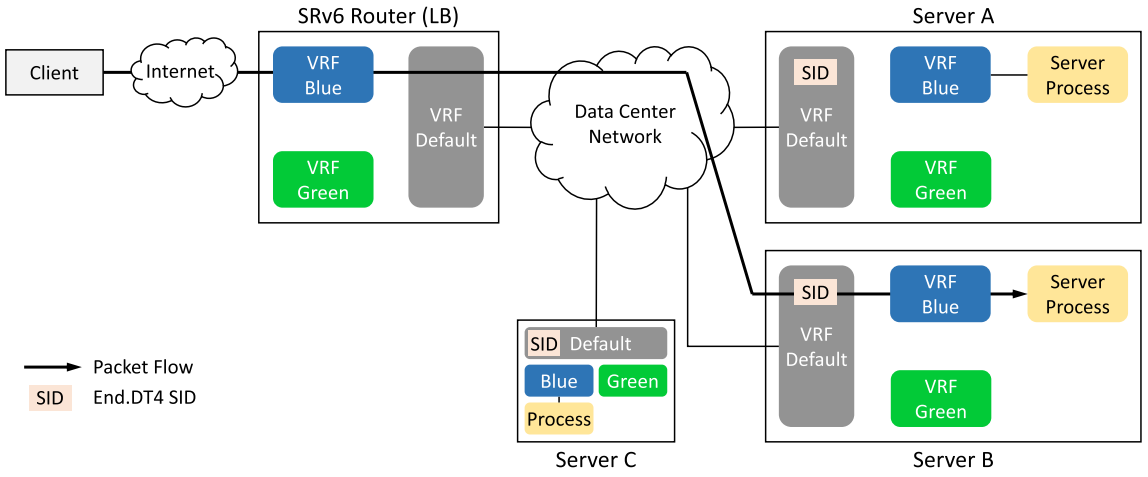
B. VRF-SHADOWING

VRF-shadowing is designed for networks composed of SRv6-based L3VPN. Figure 3a illustrates a simplified example of data center networks utilizing SRv6, where a router and three servers—Server A, B, and C—form an SRv6 L3VPN domain. The servers run server processes, e.g., HTTP servers, within VRF Blue. All servers advertise an identical IP address (VIP) in VRF Blue to the router via BGP. Additionally, the router advertises a default route of 0.0.0.0/0 in VRF Blue. Consequently, the router has an ECMP route toward the VIP via the three servers' End.DT4 SIDs as next-hops, and the servers have a default route in VRF Blue toward the router. Figure 3a also shows that a flow directed to the VIP is routed to Server B via ECMP. This scenario exemplifies a basic setup of SRv6 L3VPN in data center networks. While SRv6 is an emerging routing architecture, SRv6 has already been implemented in commodity-off-the-shelf routers and the Linux kernel. Further, there has already been a deployment of SRv6 at data centers in production [25].

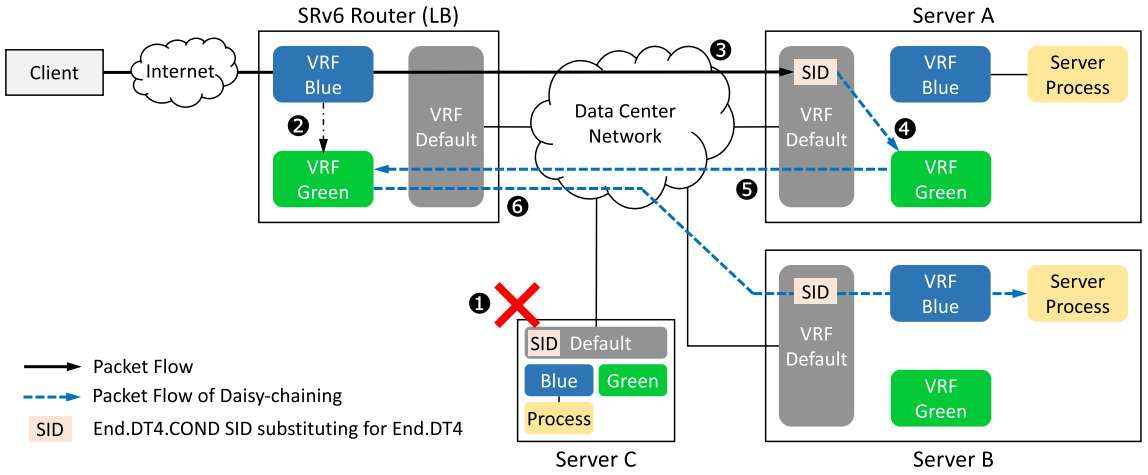
VRF-shadowing introduces two key techniques into SRv6 L3VPN to accomplish daisy-chaining without the need for additional control planes:

1) **Shadow VRF**: A VRF that retrains the previous state of routing table entries in another VRF.
2) **End.DT4.COND**: A variant of End.DT4 that submits stray packets to a different VRF.

While prior studies achieved daisy-chaining by embedding previous stats of connection-to-DIP mapping into packets, our approach maintains these previous states within a VRF (Shadow VRF) in a router—the router copies previous



(a) Simplified example of SRv6 L3VPN in data center networks. Server A, B, and C, advertise an identical IP address (VIP) in VRF Blue over BGP, and the SRv6 router has an ECMP route to the VIP with the three next-hops of End.DT4 SIDs for the servers in VRF Blue.



(b) When the next-hops change due to, for example, the removal of Server C, daisy-chaining occurs for stray packets: End.DT4.COND SID substituting for End.DT4 at Server A submits the received stray packets to VRF Green instead of VRF Blue (❹), and the packets are delivered to Server B through the previous next-hops in VRF Green at the router (❺ and ❻).

FIGURE 3. How VRF-shadowing performs daisy-chaining in an SRv6 L3VPN-based data center network.

next-hops of ECMP routes in a VRF into a Shadow VRF. End.DT4.COND at servers redirects stray packets back to the router instead of receiving them. The router then forwards the packets with the previous state in the Shadow VRF. Next, we describe how they accomplish daisy-chaining, and Section III-C and III-D will provide more detailed descriptions of Shadow VRF and End.DT4.COND.

Figure 3b illustrates how both techniques achieve daisy-chaining as a continuation of Figure 3a. Note that the End.DT4 SIDs in Figure 3a are substituted with End.DT4.COND SIDs. ❶ When a backend server, Server C in this example, is removed due to maintenance or failure, the VIP advertisement from Server C stops. ❷ At this time, the SRv6 router copies the ECMP route with the three next-hops (Server A, B, and C) from VRF Blue to VRF Green, which is configured as the Shadow VRF for VRF Blue, and then updates the next-hops for the ECMP route in VRF Blue (removing the next-hop to Server C). Consequently, VRF Green—the Shadow VRF of VRF Blue—retains the previous next-hops of the ECMP route in VRF Blue. ❸ Next, the router misdirects subsequent packets of an existing connection, intended for Server B, to the wrong server, Server A, due to the change in next-hops for the ECMP route in VRF Blue. In other words, the packets start straying.

❹ Server A executes End.DT4.COND: it receives packets to a configured VRF (VRF Blue) as with End.DT4, but if the packets are straying, it receives them to another VRF (VRF Green). ❺ As VRF Green in the servers has a default route toward VRF Green in the router, either by configuration or a route advertisement, the packets are delivered to the

router again, but this time to VRF Green. ❻ According to the Shadow VRF mechanism, VRF Green in the router retains the previous next-hops (Server A, B, and C) for the ECMP route. Therefore, the packets are forwarded again, but to the SID associated with VRF Blue on Server B, the correct server.

The above example described in Figure 3 illustrates a connection unrelated to the removed server. If a server crashes or is removed unexpectedly, connections terminated by that server are disrupted, but other connections continue through the daisy-chaining mechanism of VRF-shadowing. On the other hand, for server removal, the server should stop advertising the VIP. The router stops directing new connections to this server because there is no next-hop for the removed server in VRF Blue. Existing connections terminated by the server continue to be routed to the server through daisy-chaining via VRF Green. Once all the connections to this server end, the server can be removed without disruption.

VRF-shadowing is applicable to the assets of standardized SRv6. Shadow VRF operates independently, installing routes into a VRF regardless of the route structures. In other words, the packet format and structure of the routing table entries remain consistent with conventional SRv6. Thus, VRF-shadowing can leverage the existing packet-forwarding hardware of routers without modifications. End.DT4.COND, while being a new End behavior for SRv6, is implemented on servers and does not require external interactions. Therefore, there is no need for specialized control plane systems beyond the standardized BGP for SRv6 L3VPN to operate ECMP as a stateless and per-connection-consistent load balancer.

C. SHADOW VRF AT ROUTERS

Shadow VRF is the tiny enhancement required at the router side for implementing VRF-shadowing. As described in the previous section, Shadow VRF is a mechanism that copies routes with their associated next-hops in one VRF to another VRF (the shadow VRF) prior to updates to the next-hops in the original VRF. Consequently, the previous next-hops are preserved in the shadow VRF. This feature operates independently within routers, regardless of the routing protocols that maintain routes in the original VRF.

When implementing the Shadow VRF mechanism, two consideration points arise. The first is to update the routing table entries in the shadow VRF *before* updating the entries in the original VRF. If updating entries in the original VRF before the shadow, stray packets redirected from servers would temporarily be routed to incorrect servers until the previous next-hops are copied into the shadow VRF, potentially creating micro-loops. The second consideration is that the router must select an identical next-hop from the given next-hop entries for a flow. If the decision process for selecting a next-hop for a flow depends on factors other than next-hop entries (for example, the order of next-hop insertion and deletion), the shadow VRF might forward packets to a different next-hop, even though the shadow VRF has next-hops identical to the previous ones in the original VRF.

D. END.DT4.COND AT SERVERS

End.DT4.COND is responsible for shifting stray packets to the shadow VRF instead of the original VRF, in addition to performing the functions of End.DT4. End.DT4.COND changes the VRF for receiving packets by a condition: whether the receiving packets are straying. Algorithm 1 outlines the simplified procedure of End.DT4.COND.[2] End.DT4.COND processes every received packet destined for an associated local SID, with two configured VRFs—*original* and *backup* (corresponding to VRF Blue and Green at the servers in Figure 3b, respectively). First, it decapsulates the received packet and then determines whether the inner packet is part of an existing connection. If the packet is a TCP packet and the SYN flag is not set, the packet is a part of a connection. If so, it further checks if the node is terminating the connection by searching for a socket that matches the 5-tuple of the packet. If no matching sockets are found, this indicates that another node is terminating the connection and the packet is straying. End.DT4.COND receives the stray packet to the *backup* VRF, which then redirects the packet to the router by a default route installed in the *backup* VRF. Otherwise (the packet is not TCP, it is the beginning of a new flow, or the node terminates the connection), End.DT4.COND receives the packet to the *original* VRF. Server processes running on the *original* VRF will consume the packet.

Algorithm 1 Procedure of End.DT4.COND

```
1: function EndDT4Cond(packet, original, backup)
2:     innerPacket ← decapsulateSRH(packet)
3:     if innerPacket is TCP and SYN flag is not set then
4:         if findSocket(innerPacket) is None then
5:             return receiveToVRF(innerPacket, backup)
6:         end if
7:     end if
8:     return receiveToVRF(innerPacket, original)
9: end function
```

Note that, for simplicity, we have described End.DT4.COND for IPv4 in SRv6 packets so far. The procedure applies equally to IPv6 in SRv6 packets (End.DT6.COND) and to both IPv4 and IPv6 in SRv6 packets for a VRF (End.DT46.COND).

IV. LIMITATIONS AND DEPLOYMENT CONSIDERATIONS

VRF-shadowing has some limitations derived from ECMP and issues due to daisy-chaining. This section briefly discusses these considerations.

[2] An SRv6 End behavior must perform regular processing, e.g., validating SRH and ICMP handling, as described in RFC8986. Here, we focus on the logic of End.DT4.COND and omit those details for the sake of clarity.

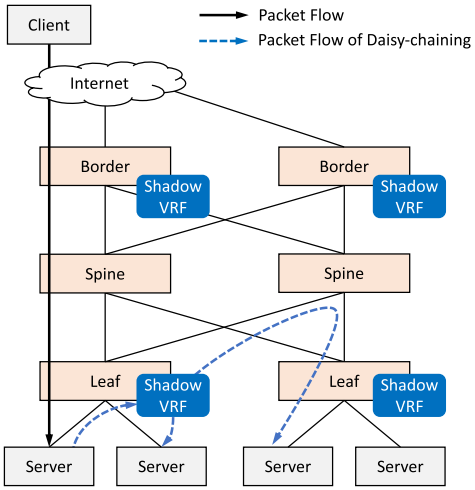


FIGURE 4. An anycast End.DT4 SID for shadow VRFs on the border and leaf switches localizes redirected traffic.

```
ip vrf blue
 shadow green
 rd 1:100
 route-target import 100:100
 route-target export 100:100
 segment-routing srv6 locator loc
exit
!
ip vrf green
 rd 1:101
 route-target import 101:101
 route-target export 101:101
 segment-routing srv6 locator loc
exit
```

FIGURE 5. An example configuration for Shadow VRF in FX2. The `shadow` attribute in `vrf blue` specifies that `green` is its Shadow VRF.

A. EFFICIENCY OF LOAD DISTRIBUTION

ECMP can lead to load imbalance between next-hops due to factors such as flow bandwidth-agnostic traffic splitting [26], [27], [28], [29]. Meanwhile, numerous efforts have been made to improve load distribution efficiency in ECMP. Adopting these techniques would address the imbalance separately from the PCC provided by VRF-shadowing. Weighted cost multi-path has long been studied [30], [31], [32]. From a practical perspective, BGP has the Link Bandwidth extended community [33] to enable weighted traffic splitting over BGP multi-path. Cisco routers implement this extended community [34], and Nvidia also supports it for data center use, called W-ECMP [35]. A similar extended community for weighted load balancing has been proposed for Ethernet VPN [36]. The concept of Shadow VRF, which stores the previous state in a separate VRF for daisy-chaining, is independently applicable to these weighted traffic splitting mechanisms.

B. ASSUMING CONNECTION-ORIENTED TRANSPORT PROTOCOLS

VRF-shadowing can handle only connection-oriented protocols. End.DT4.COND must identify an associated connection for each packet to determine the appropriate receiving VRF. While Algorithm 1 focuses on TCP, other protocols, such as QUIC, are also applicable from the protocol viewpoint. However, an implementation challenge arises with QUIC. QUIC is typically implemented in user space. If End.DT4.COND operates at the kernel-level network stack, a mechanism is needed for the QUIC user-space implementation to communicate connection information to the kernel.

C. LOCALIZING REDIRECTED TRAFFIC

As depicted in Figure 3b, all stray packets redirected by End.DT4.COND pass through the SRv6 router, which can lead to inefficient bandwidth usage depending on topologies. Utilizing anycast SID [21] can localize this redirected traffic in typical data center networks such as spine-leaf topologies. An anycast SID (or anycast segment) is a single SID advertised from multiple SR nodes. As with IP anycast, traffic destined for the anycast SID is steered to the closest node advertising the SID. Figure 4 illustrates how daisy-chain traffic can be localized using nycast SID. The border routers and leaf switches are part of the same SRv6 L3VPN domain; therefore, they have identical routing table entries in the VRFs. Next, the border routers and leaf switches have and advertise an identical End.DT4 SID (anycast SID) associated with their shadow VRFs. As a result, redirected packets encapsulated in SRv6 are routed to the nearest leaf switch advertising the anycast SID and then forwarded based on the previous next-hops in the shadow VRF, as depicted as dashed arrows in Figure 4.

V. IMPLEMENTATION

We implemented Shadow VRF on a production router, FITELNet FX2 from FURUKAWA Electric [37], which is equipped with 24 25Gbps ports and four 100Gbps ports. FX2 already supported SRv6 L3VPN; thus, we added the Shadow VRF feature with only 109 steps of codes. Figure 5 shows an example of VRF configurations that include Shadow VRF. Here, two VRFs, `blue` and `green`, are defined and configured for SRv6 L3VPN, for example, route distinguisher and route targets. The `shadow` attribute in VRF blue designates VRF green as its shadow VRF, so the previous next-hops of routes in VRF blue will be copied to VRF green.

We also implemented End.DT4.COND for Linux with eBPF [16]. eBPF is a virtual instruction set, and Linux provides various hook points to execute eBPF programs, such as Traffic Control (tc) infrastructure for receiving and transmitting packets [38]. Figure 6 shows the operation of the eBPF implementation of End.DT4.COND. The eBPF program is attached to a network interface via tc, represented by the diamond shape in the figure. It parses the headers of a received packet and checks whether the packet is encapsulated in SRv6. If so, it checks whether the inner

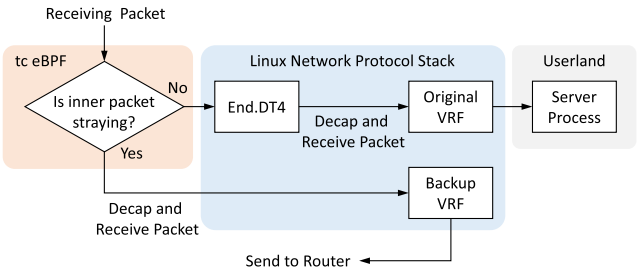


FIGURE 6. The flow diagram of the End.DT4.COND implementation in eBPF.

packet is part of a connection being terminated by another node (straying) as per Algorithm 1. If the inner packet is straying, it is then directed to a specified (*backup*) VRF. Otherwise, packets not identified as straying are processed by the kernel network protocol stack, namely, the original End.DT4, resulting in the packets being received in the original VRF where the server process runs. The End.DT4.COND eBPF program has counters for received and redirected packets, and it is written in C with just 340 lines of code.

The Linux network protocol stack supports various SRv6 End behaviors as routing table entries by `seg6local` infrastructure [39]; however, we implemented End.DT4.COND in tc eBPF and not in `seg6local`. The advantage of our design choice is that End.DT4.COND works independently of other SRv6 routes managed by other components. Let us consider a Linux node participating in an SRv6 L3VPN domain. `bgpd` of FRRouting [40], a popular routing suite supporting SRv6 L3VPN, installs End.DT4 routes for each local VRF. End.DT4.COND in eBPF attached to physical interfaces can coexist with these End.DT4 routes. If the received packets are straying, End.DT4.COND receives the packets to a backup VRF; otherwise, the End.DT4 route processes the packets. This design adds the End.DT4.COND functionality without disturbing existing SRv6 implementations of the Linux kernel and FRRouting. Implementing End.DT4.COND as `seg6local` routing table entries like End.DT4 would have required modifications to FRRouting to install End.DT4.COND routes instead of End.DT4 routes.

We slightly modified the Linux kernel to address an L3VPN-specific issue for searching for sockets. The End.DT4.COND implementation uses bpf-helpers, which can invoke kernel functionalities from eBPF programs [41], to find a corresponding socket for an inner packet (line 4 in Algorithm 1). The function we used is `bpf_skc_lookup_tcp()`, which returns a that matches 5-tuple values passed to the function or NULL if not found. This function searches for sockets associated with the physical interface that receives the packet. On the other hand, in Linux, a VRF is represented as a virtual interface (called l3mdev [42]), and processes running on a VRF receive packets from the l3mdev associated with the VRF. In the case of SRv6 L3VPN, packets decapsulated by End.DT4 are received from the l3mdev interface corresponding to the VRF. Therefore, `bpf_skc_lookup_tcp()` called for packets received by a physical interface cannot find sockets opened within VRFs. To enable tc eBPF programs attached to physical interfaces to search for sockets associated with other interfaces, we added five lines of code that introduced a new flag to specify an interface to use as the search key.

VI. EVALUATION

In this section, we evaluate VRF-shadowing focusing on three aspects:

1) Does VRF-shadowing perform as expected?
2) How resilient is VRF-shadowing to next-hop changes?
3) Overhead due to End.DT4.COND at the server side.

Since VRF-shadowing leverages the existing data planes of routers supporting SRv6, we do not evaluate the packet forwarding performance of the load balancer. Instead, our evaluation focuses on the behavior of daisy-chaining by VRF-shadowing on the first and second aspects. In addition, we investigate the performance degradation on the server side due to the End.DT4.COND eBPF program.

A. EXPERIMENTAL ENVIRONMENT

We prepared an experimental environment comprising two physical server machines and an SRv6 router. Figure 7 illustrates the environment. The two server machines were identical: DELL PowerEdge R440 equipped with an Xeon Gold 6130 16-core CPU, 48GB memory, and an Nvidia ConnectX-6 Dx 100Gbps NIC. They run Ubuntu 22.04.1 with Linux kernel 5.15.0. Both machines were connected to the router—FX2 having the Shadow VRF feature—with 100Gbps links.

In the physical topology shown on the left side of Figure 7, we emulated eight independent HTTP servers behind FX2, which performed a load balancer, as depicted on the right side of the figure. We deployed eight containers on one of the servers (the bottom one in the figure); each container had a virtual network interface created by SR-IOV and ran FRRouting and H2O HTTP server [43]. All SR-IOV virtual interfaces and their associated physical interface were connected to an embedded switch in the NIC board, allowing the containers to act as eight independent hosts connected to the load balancer. The load balancer and the eight containers used OSPFv3 to exchange the underlying IPv6 addresses and BGP to compose an SRv6 L3VPN domain. Moreover, all containers advertised the same IP address in a VRF where the H2O processes were running. The IP address performed a VIP for the eight HTTP servers. The load balancer installed a routing table entry in the VRF, with the destination being the VIP and the next-hops being End.DT4 SIDs of the eight containers. Finally, the Shadow VRF feature was configured and enabled on the load balancer, and the End.DT4.COND implementation was attached to the virtual interfaces of all containers.

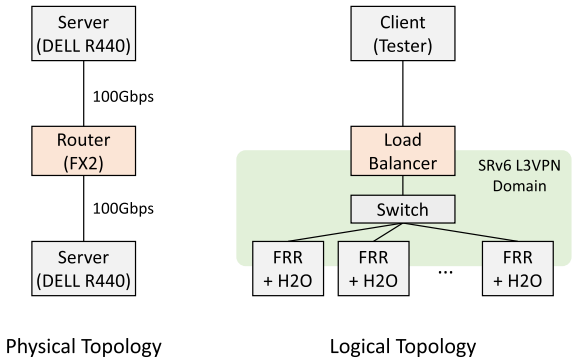


FIGURE 7. Experimental environment.

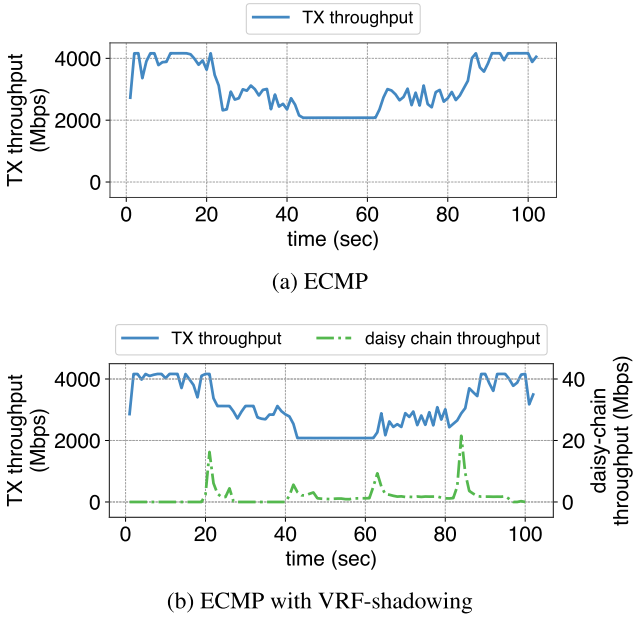


FIGURE 8. The transition of throughput while removing and adding servers step-by-step per 20 seconds.

Another server (the tester machine) emulated clients issuing HTTP GET requests to the VIP using Apache Bench (`ab`) [44]. Note that we slightly modified `ab`. The original version stops benchmarking when an HTTP GET request times out, whereas our modified version continues even when requests time out or fail. This modification allows the experiment to proceed when ECMP disrupts the connections.

We also attached an eBPF program to the tester machine to emulate multiple source IP addresses. `ab` cannot use multiple IP addresses as sources; therefore, HTTP GET requests to the VIP generated by `ab` have the same source IP address (assigned to the tester machine) and destination IP address (VIP). This environment-specific situation hinders the potential for traffic distribution by ECMP on FX2 due to its hash calculation logic. The eBPF program we implemented was attached to the tester machine's interface, and it manipulated outbound and inbound packets in a stateless manner. The program embeds the source port number into the lower 16 bits of the source IP address field for outbound packets, and it restores the original IP address in the destination IP address field for inbound packets. This approach allowed us to emulate 2^{16} different source IP addresses over a single IP address assigned to the machine.

All components in the environment—the eight HTTP servers, the load balancer, and the tester machine—were connected with the 100Gbps links. Thus, the bottleneck in this setup was the throughput of HTTP GET requests that the single tester machine running `ab` can generate, compared with the traffic that parallelized HTTP servers can produce. In other words, adding more HTTP servers does not increase the total throughput because `ab` is the bottleneck. To emulate a significant volume of client requests, we limited the bandwidth of the virtual interfaces of the containers to 1Gbps. This bandwidth limitation is intended to emulate many requests that a single server cannot process alone.

B. DOES VRF-SHADOWING PERFORM AS EXPECTED?

The first experiment is to confirm whether VRF-shadowing performs as expected. Initially, there were four HTTP servers (containers), and the tester machine started to issue 64 concurrent HTTP GET requests for a 10M-byte file to the VIP. Then, two HTTP servers were temporarily removed by stopping their VIP advertisement at 20 and 40 seconds and then restored by restarting the VIP advertisement at 60 and 80 seconds. During the experiment, we measured the throughput sent by HTTP servers and counted the failed HTTP GET requests recorded by `ab`.

Figure 8 shows the transition of the throughput sent by the HTTP servers. Traditional ECMP (without VRF-shadowing) is shown in Figure 8a, and the case with VRF-shadowing is shown in Figure 8b. Figure 8b also plots the throughput of daisy-chain traffic measured by the End.DT4.COND eBPF program attached to the virtual interfaces of the HTTP servers. As shown in both cases, there are no significant differences in request distributions and the resulting throughput, whether with or without VRF-shadowing; the throughput started at approximately 4Gbps—1Gbps per HTTP server—decreased to 2Gbps and returned to 4Gbps as the servers were removed and added back.

However, there was a noticeable difference in the rate of failed HTTP GET requests between those with and without VRF shadowing. Traditional ECMP experienced a 9.8% failure rate (381 out of 3902 requests), while VRF-shadowing had a 0% failure rate (0 out of 3819 requests). This result suggests that VRF-shadowing provides PCC for ECMP through daisy-chaining as expected; it protects connections from disruption to next-hop changes in ECMP. The spikes in the daisy-chain traffic throughput in Figure 8b also indicate that daisy-chaining occurred when the servers were removed and restored.

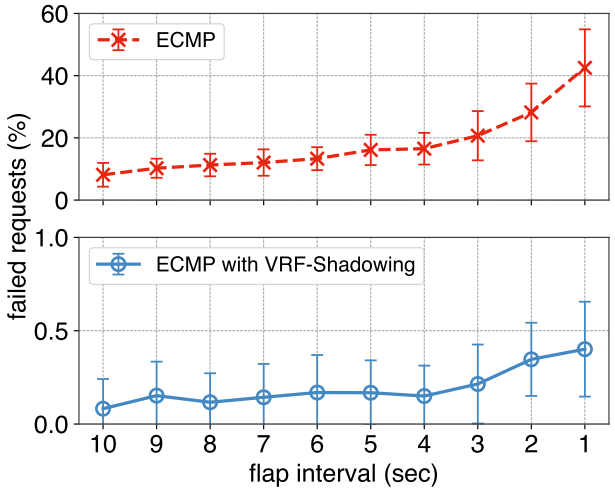


FIGURE 9. Rate of failed HTTP GET requests (5MB file).

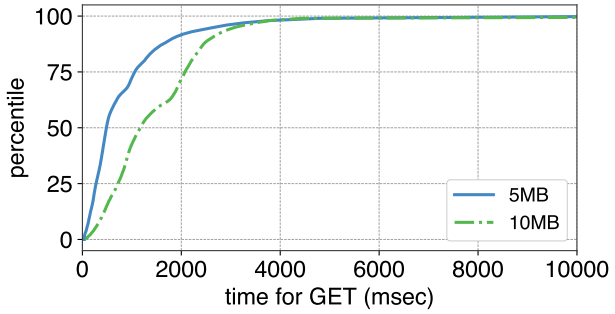


FIGURE 10. CDF of times for completing HTTP GET requests for a file of 5MB or 10MB by `ab`.

C. HOW RESILIENT IS VRF-SHADOWING FOR CHURNS?

Next, we evaluate the ability of VRF-shadowing to maintain PCC amid changes in the number of next-hops. In this experiment, `ab` issued HTTP GET requests to the VIP for 30 seconds while the eight HTTP servers flapped; a randomly selected server went down (withdrawing the VIP) or up (advertising the VIP) at fixed intervals.

Figure 9 shows the result when `ab` retrieved a 5MB file. The x-axis is the server flap interval, and the y-axis indicates the rate of failed HTTP GET requests reported by `ab`. We conducted 30 runs for each server flap interval, ranging from 1 to 10 seconds, and the y-axis shows the average failure rate and their standard deviation over the 30 runs. As shown, traditional ECMP disrupts approximately 8% of requests when next-hops change every 10 seconds. This failure rate increases up to 42.5% as the interval is shortened. Every next-hop change in the ECMP route disperses existing flows to different servers, leading to frequent connection disruptions, particularly with shorter flap intervals.

In contrast, VRF-shadowing achieves a significantly lower failure rate ($<$ 1%). However, the failure rate is not zero—ranging from 0.08% to 0.4% as the interval shortens. This non-zero failure rate is attributed to the nature of

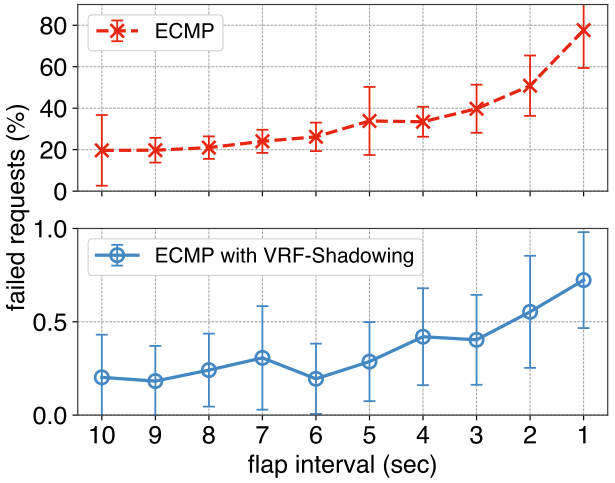


FIGURE 11. Rate of failed HTTP GET requests (10MB file).

daisy-chaining, which redirects stray packets to the *last* server. When different next-hops are selected twice for a continuous connection due to two consecutive next-hop changes, daisy-chaining fails to deliver the connection to the appropriate server, resulting in disruption. To confirm this point, we measured the time required to complete HTTP GET requests by issuing 10,000 requests with `ab` in the experimental environment. Figure 10 shows the CDF of the times for completing the HTTP GET requests. For a 5MB file, 8% of the GET requests took longer than 2 seconds to complete, with the longest connection taking over 10 seconds (11.8 seconds). Disruptions could occur for such long-lived connections if two next-hop changes happen, even with VRF-shadowing.

Long-lived connections are expected to be more susceptible to disruption due to next-hop changes. We conducted the same experiment except that ab retrieved a 10MB file to verify this point. The dashed green line in Figure 10 shows the times to complete the HTTP GET requests for a 10MB file. As expected, it takes longer than getting the 5MB file. Figure 11 shows the failure rate versus flap intervals for the 10MB file scenario. As with Figure 9, we conducted 30 runs for each server flap interval, and the y-axis shows the average failure rate and their standard deviation over the 30 runs. As shown, the failure rate of the traditional ECMP doubles compared to the results in Figure 9. For instance, 77% of GET requests failed when the flap interval was one second. Although VRF-shadowing also increases the failure rate for longer-lived connections, the rate of failed requests remains low, ranging from 0.2% to 0.7% across the flap intervals, compared with the traditional ECMP.

Next, we examined the amount of redirected traffic from the servers to for daisy-chaining. Figure 12 shows the average ratio of the redirected traffic to the received traffic measured by the eBPF End.DT4.COND program at the servers during the experiments of Figure 9 and 11. This ratio increases along with the frequency of next-hop changes. Also,

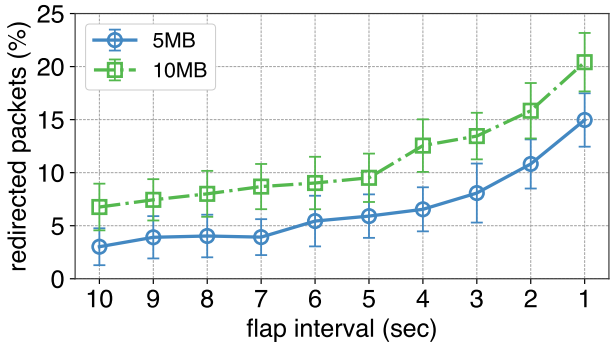


FIGURE 12. Ratio of redirected packets from the servers to the router by VRF-shadowing.

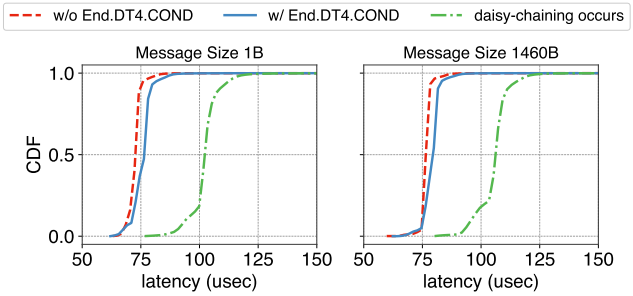


FIGURE 13. Latency overhead due to End.DT4.COND and daisy-chaining.

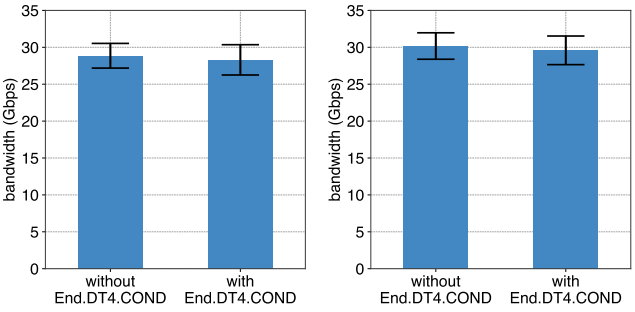


FIGURE 14. Throughput with/without End.DT4.COND.

longer connections typically involve more packets needing redirection until the connections end. Therefore, getting the 10MB file resulted in a higher redirected ratio than the 5MB case. Specifically, the peak redirection ratio is 20.4% with a flap interval of one second while getting the 10MB file. Note that redirected traffic in daisy-chaining consists of packets sent from clients to servers, e.g., requests and ACKs. Such incoming traffic is generally smaller in volume than the outgoing traffic from the servers. In our experiment, the peak throughput of redirected traffic from a server is just 5.92Mbps.

Based on these results, we conclude that VRF-shadowing effectively implements daisy-chaining to preserve PCC on ECMP over SRv6 L3VPN without additional control plane systems. Daisy-chaining does not guarantee complete PCC; nevertheless, it remains an effective solution for practical operations [1], [2]. VRF-shadowing brings daisy-chaining to SRv6 L3VPN with only standardized routing protocols.

D. OVERHEAD AT THE SERVER SIDE

So far, we have observed the daisy-chaining aspect of VRF-shadowing. We next measured overhead due to End.DT4.COND on the server side. To clarify the overhead on latency, we implemented a simple server-client program that measures round-trip times over a TCP connection; the client sends a message, and the server echoes it back. The server process ran inside a container as with the HTTP servers in the previous experiments, and we ran the client program at the tester machine. With this setup, we measured the latency in three scenarios: without the End.DT4.COND eBPF program, with it attached, and while daisy-chaining occurs.

Figure 13 shows the CDF of 100,000 measured round-trip times for each scenario, with one and 1460-byte message sizes. The results indicate that attaching End.DT4.COND adds several microseconds of latency, and daisy-chaining introduces an additional 15–34 microseconds. The former is attributed to the processing time of the eBPF program, and the latter involves sending the packet to the router and forwarding it with the previous next-hop. This microsecond-scale latency would be considerable for transactions demanding ultra-low latency inside a data center [45], but it is still acceptable for client-server communications over the Internet.

In the throughput experiment, we directly connected the two server machines via a 100Gbps link. The servers formed an SRv6 L3VPN domain using FRRouting. An iperf3 server process ran in a VRF on a server, with or without the End.DT4.COND eBPF program, and another server sent or received TCP traffic with an iperf3 client. Figure 14 shows the average and standard deviations of the transmitting and receiving throughput over 30 runs in each configuration. The results reveal no significant differences in the throughput with and without End.DT4.COND. Thus, the processing overhead for End.DT4.COND is not significant, thanks to the Linux eBPF infrastructure.

VII. CONCLUSION

In this paper, we have proposed VRF-shadowing that achieves daisy-chaining for PCC in the SRv6 L3VPN framework without the need for additional control plane systems. The SRv6 router retains the previous states of ECMP next-hops in a separate VRF, and the servers redirect stray packets to the correct servers through the VRF by a new SRv6 End behavior. As a result, the SRv6 router can perform load balancers that support PCC with existing packet-forwarding planes and standardized routing protocols. We have implemented VRF-shadowing with a commercial router and Linux. The evaluation demonstrates that VRF-shadowing prevents connection disruptions when ECMP next-hops change. The most severe case shows that

VRF-shadowing reduces the rate of failed requests to 0.7%, in contrast to ECMP, which disrupts 77% of requests due to next-hop changes. Although daisy-chaining does not ensure complete PCC and naive ECMP has limited balancing performance, VRF-shadowing offers a practical solution for operational simplicity in deploying hardware load balancers.

REFERENCES

[1] J. T. Araujo, L. Saino, L. Buytenhek, and R. Landa, "Balancing on the edge: Transport affinity without network state," in *Proc. 15th USENIX Symp. Networked Syst. Design Implement.*, Apr. 2018, pp. 111–124.

[2] D. Wragg. (2020). *Unimog—Cloudflare's Edge Load Balancer*. [Online]. Available: https://blog.cloudflare.com/unimog-cloudflares-edge-load-balancer

[3] D. E. Eisenbud, C. Yi, C. Contavalli, C. Smith, R. Kononov, E. Mann-Hielscher, A. Cilingiroglu, and B. Cheyney, and W. Shang, "Maglev: A fast and reliable software network load balancer," in *Proc. 13th Symp. Netw. Syst. Design Implement.*, 2016, pp. 523–535.

[4] R. Miao, H. Zeng, C. Kim, J. Lee, and M. Yu, "SilkRoad: Making stateful layer-4 load balancing fast and cheap using switching ASICs," in *Proc. Conf. ACM Special Interest Group Data Commun.*, Aug. 2017, pp. 15–28.

[5] C. Zeng, L. Luo, T. Zhang, Z. Wang, L. Li, W. Han, N. Chen, L. Wan, L. Liu, Z. Ding, X. Geng, T. Feng, F. Ning, K. Chen, and C. Guo, "Tiara: A scalable and efficient hardware acceleration architecture for stateful layer-4 load balancing," in *Proc. 19th USENIX Symp. Netw. Syst. Design Implement. (NSDI)*, 2022, pp. 1345–1358.

[6] P. Patel, D. Bansal, L. Yuan, A. Murthy, A. Greenberg, D. A. Maltz, R. Kern, H. Kumar, M. Zikos, H. Wu, C. Kim, and N. Karri, "Ananta: Cloud scale load balancing," in *Proc. ACM SIGCOMM Conf.*, Aug. 2013, pp. 207–218.

[7] R. Gandhi, H. H. Liu, Y. C. Hu, G. Lu, J. Padhye, L. Yuan, and M. Zhang, "Duet: Cloud scale load balancing with hardware and software," in *Proc. ACM Conf. SIGCOMM*. New York, NY, USA: Association for Computing Machinery, Aug. 2014, pp. 27–38, doi: 10.1145/2619239.2626317.

[8] C. Hopps. (2000). *Analysis of an Equal-Cost Multi-Path Algorithm*. [Online]. Available: https://www.rfc-editor.org/info/rfc2992

[9] W. Zhang. (2016). *IPVS Software—Advanced Layer-4 Switching*. [Online]. Available: http://www.linuxvirtualserver.org/software/ipvs.html

[10] (2023). *Facebookincubator/Katran: A High Performance Layer 4 Load Balancer*. [Online]. Available: https://github.com/facebookincubator/katran

[11] (2023). *Github Load Balancer Director and Supporting Tooling*. [Online]. Available: https://github.com/github/glb-director

[12] (2023). *Haproxy—The Reliable, High Performance TCP/HTTP Load Balancer*. [Online]. Available: http://www.haproxy.org/

[13] R. Cohen, M. Kadosh, A. Lo, and Q. Sayah, "LB scalability: Achieving the right balance between being stateful and stateless," *IEEE/ACM Trans. Netw.*, vol. 30, no. 1, pp. 382–393, Feb. 2022, doi: 10.1109/TNET.2021.3112517. https://doi.org/10.1109/TNET.2021.3112517

[14] V. Olteanu, A. Agache, A. Voinescu, and C. Raiciu, "Stateless datacenter load-balancing with beamer," in *Proc. 15th USENIX Symp. Networked Syst. Design Implement. (NSDI)*, 2018, pp. 125–139.

[15] R. Nakamura, K. Ebisawa, T. Okuzawa, C. Lee, and Y. Sekiya, "Extending ecmp toward a practical hardware load balancer," in *Proc. Internet Operation Technol. Symp.*, Dec. 2022, pp. 40–47.

[16] kernel Develop. communit. *BPF Documentation—The Linux Kernel Documentation*. Accessed: Jun. 13, 2024. [Online]. Available: https://docs.kernel.org/bpf/

[17] T. Barbette, C. Tang, H. Yao, D. Kostić, G. Q. Maguire, P. Papadimitratos, and M. Chiesa, "A high-speed load-balancer design with guaranteed per-connection-consistency," in *Proc. 17th USENIX Symp. Networked Syst. Design Implement. (NSDI)*, 2020, pp. 667–683.

[18] B. Pit-Claudel, Y. Desmouceaux, P. Pfister, M. Townsley, and T. Clausen, "Stateless load-aware load balancing in P4," in *Proc. IEEE 26th Int. Conf. Netw. Protocols (ICNP)*, Sep. 2018, pp. 418–423.

[19] Y. Desmouceaux, P. Pfister, J. Tollet, M. Townsley, and T. Clausen, "6LB: Scalable and application-aware load balancing with segment routing," *IEEE/ACM Trans. Netw.*, vol. 26, no. 2, pp. 819–834, Apr. 2018.

[20] R. Gandhi, Y. C. Hu, C. kok Koh, H. H. Liu, and M. Zhang, "Rubik: Unlocking the power of locality and end-point flexibility in cloud scale load balancing," in *Proc. USENIX Annu. Tech. Conf.*, Jul. 2015, pp. 473–485.

[21] C. Filsfils, S. Previdi, L. Ginsberg, B. Decraene, S. Litkowski, and R. Shakir. (2018). *Segment Routing Architecture*. [Online]. Available: https://www.rfc-editor.org/info/rfc8402

[22] C. Filsfils, D. Dukes, S. Previdi, J. Leddy, S. Matsushima, and D. Voyer. (2020). *IPV6 Segment Routing Header (SRH)*. [Online]. Available: https://www.rfc-editor.org/info/rfc8754

[23] C. Filsfils, P. Camarillo, J. Leddy, D. Voyer, S. Matsushima, and Z. Li. (2021). *Segment Routing Over IPV6 (SRV6) Network Programming*. [Online]. Available: https://www.rfc-editor.org/info/rfc8986

[24] G. Dawra, K. Talaulikar, R. Raszuk, B. Decraene, S. Zhuang, and J. Rabadan, *BGP Overlay Services Based on Segment Routing Over IPV6 (SRV6)*, document RFC 9252, 2022.

[25] S. Matsushima, C. Filsfils, Z. Ali, Z. Li, K. Rajaraman, and A. Dhamija. (2022). *SRV6 Implementation and Deployment Status*. [Online]. Available: https://datatracker.ietf.org/doc/draft-matsushima-spring-srv6-deployment-status/15/

[26] 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.

[27] Y. Cao and M. Xu, "Dual-NAT: Dynamic multipath flow scheduling for data center networks," in *Proc. 21st IEEE Int. Conf. Netw. Protocols (ICNP)*, Oct. 2013, p. 19.

[28] Z. Zhang, H. Zheng, J. Hu, X. Yu, C. Qi, X. Shi, and G. Wang, "Hashing linearity enables relative path control in data centers," in *Proc. USENIX Annu. Tech. Conf.*, Jul. 2021, pp. 855–862.

[29] Y. Xu, K. He, R. Wang, M. Yu, N. Duffield, H. Wassel, S. Zhang, L. Poutievski, J. Zhou, and A. Vahdat, "Hashing design in modern networks: Challenges and mitigation techniques," in *Proc. USENIX Annual Technical Conf.*, 2022, pp. 805–818.

[30] J. Zhou, M. Tewari, M. Zhu, A. Kabbani, L. Poutievski, A. Singh, and A. Vahdat, "WCMP: Weighted cost multipathing for improved fairness in data centers," in *Proc. 9th Eur. Conf. Comput. Syst.*, Apr. 2014, pp. 1–14.

[31] M. Shafiee and J. Ghaderi, "A simple congestion-aware algorithm for load balancing in datacenter networks," *IEEE/ACM Trans. Netw.*, vol. 25, no. 6, pp. 3670–3682, Dec. 2017.

[32] K.-F. Hsu, P. Tammana, R. Beckett, A. Chen, J. Rexford, and D. Walker, "Adaptive weighted traffic splitting in programmable data planes," in *Proc. Symp. SDN Res.*, Mar. 2020, pp. 103–109, doi: 10.1145/3373360.3380841.

[33] P. Mohapatra and R. Fernando. (2018). *BGP Link Bandwidth Extended Community*. [Online]. Available: https://datatracker.ietf.org/doc/draft-ietf-idr-link-bandwidth/07/

[34] Cisco Syst. (2016). *IP Routing: BGP Configuration Guide—BGP Link Bandwidth [Cisco ASR 1000 Series Aggregation Services Routers]*. [Online]. Available: https://www.cisco.com/c/en/us/td/docs/ios-xml/ios/iproute_bgp/configuration/xe-16/irg-xe-16-book/bgp-link-bandwidth.html

[35] NVIDIA Corp. (2024). *BGP Weighted Equal Cost Multipath | Cumulus Linux 5.9*. [Online]. Available: https://docs.nvidia.com/networking-ethernet-software/cumulus-linux-59/Layer-3/Routing/BGP-Weighted-Equal-Cost-Multipath/

[36] N. Malhotra, A. Sajassi, J. Rabadan, J. Drake, A. R. Lingala, and S. Thoria. (2023). *Weighted Multi-Path Procedures for EVPN Multi-Homing*. [Online]. Available: https://datatracker.ietf.org/doc/draft-ietf-bess-evpn-unequal-lb/21/

[37] Furukawa Electric. *List of Data Cneter Products*. Accessed: Jun. 13, 2024. [Online]. Available: https://www.furukawa.co.jp/en/solution/datacenter/product.html

[38] D. Borkmann, "On getting TC classifier fully programmable with CLS BPF," in *Proc. Tech. Conf. Linux Netw.*, Nov. 2016, pp. 1–19.

[39] (2019). *SRV6—Linux Kernel Implementation | Implementation / Advanced Configuration*. [Online]. Available: https://segment-routing.org/index.php/Implementation/AdvancedConf

[40] FRRouting Protject. (2023). *Frrouting*. [Online]. Available: https://frrouting.org/

[41] (2022). *BPF—Helpers(7)—Linux Manual Page*. [Online]. Available: https://man7.org/linux/man-pages/man7/bpf-helpers.7.html

[42] (2021). *Virtual Routing and Forwarding (VRF); The Linux Kernel Documentation*. [Online]. Available: https://docs.kernel.org/networking/vrf.html

[43] (2024). *H2O—The Optimized Http Server*. [Online]. Available: https://h2o.examp1e.net/
[44] Apache Softw. Found. (2022). *AB—Apache HTTP Server Benchmarking Tool—Apache HTTP Server Version 2.4*. [Online]. Available: https://httpd.apache.org/docs/2.4/programs/ab.html
[45] B. Montazeri, Y. Li, M. Alizadeh, and J. Ousterhout, "Homa: A receiver-driven low-latency transport protocol using network priorities," in *Proc. Conf. ACM Special Interest Group Data Commun.*, Aug. 2018, pp. 221–235, doi: 10.1145/3230543.3230564.

RYO NAKAMURA received the Ph.D. degree in information science and technology from The University of Tokyo, Tokyo, Japan, in 2017.

He is an Associate Professor with the Information Technology Center, The University of Tokyo, and operates The University of Tokyo campus network. His research interests include the networking aspect in operating systems, network virtualization, and network operation.

KENTARO EBISAWA received the degree in Earth and planetary sciences from Tokyo Institute of Technology, in 1997. He has engaged in the early stages of the internet, focusing on ADSL, FTTH, and VPN, for verification and support for service implementation. Subsequently, he worked across various functions, including customer support, product design, and development management, primarily in domestic and international startups. After contributing to the development of high-performance network products using ASICs and FPGAs, such as flow routers and OpenFlow switches, he was a Research Associate of major automobile companies and telecommunications operators, involved in the research and development of protocols and systems utilizing data plane programming (P4). He is also a co-translator, a supervisor, and the contributing author of the book: *Software-Defined Networks: Concepts, Design, and Use Cases*.

HIDEAKI HAYASHI received the master's degree in information science and technology from Tokyo University of Agriculture and Technology, Tokyo, Japan. Currently, he is a Blank Software Engineer with Furukawa Network Solution Corporation, specializing in BGP and segment routing. His research interest includes network architecture and virtualization.

TATSUYA FUJIWARA is a Software Engineer with Furukawa Network Solution Corporation. His research interests include routing architecture and network virtualization.

TOMOKO OKUZAWA has been a Senior Engineer with Toyota Motor Corporation, since 2022. Leveraging her extensive network architect background from service provider experience, she currently focuses on researching and developing digital data platforms for collecting, storing, and analyzing data generated by connected cars.